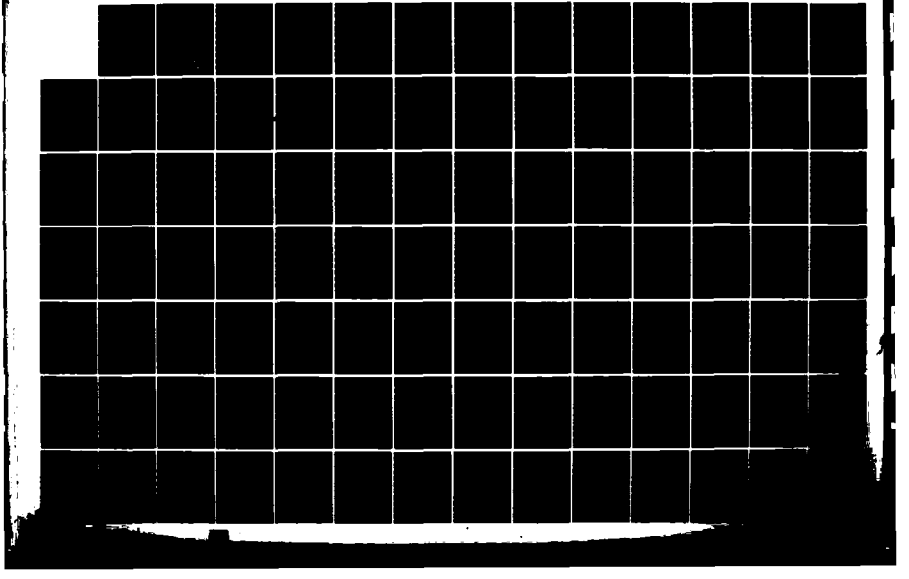
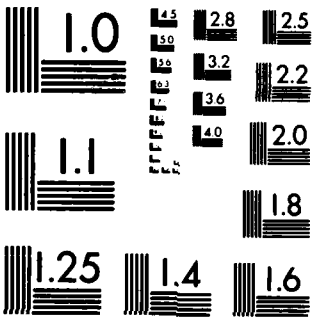
AD-A150 312 MOTION PLANNING WITH SIX DEGREES OF FREEDOM(U) 1/3
MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL
INTELLIGENCE LAB B R DONALD MAY 84 AI-TR-791
UNCLASSIFIED N00014-81-K-0494 F/G 6/4 NL





Technical Report 791

Motion Planning with Six Degrees of Freedom

Bruce R. Donald

MIT Artificial Intelligence Laboratory

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AI-TR-791	2. GOVT ACCESSION NO. AD-A150 312	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Motion Planning with Six Degrees of Freedom		5. TYPE OF REPORT & PERIOD COVERED technical report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Bruce R. Donald		8. CONTRACT OR GRANT NUMBER(s) N00014-81-K-0494 N00014-80-C-0505 N00014-82-K-0334
9. PERFORMING ORGANIZATION NAME AND ADDRESS Artificial Intelligence Laboratory 545 Technology Square Cambridge, Massachusetts 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Blvd Arlington, Virginia 22209		12. REPORT DATE MAY 1984 13. NUMBER OF PAGES 261
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research Information Systems Arlington, Virginia 22217		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Distribution is unlimited

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Motion planning, path planning, geometric planning,
configuration space, robotics, collision avoidance.
generalized Voroni diagram, spatial reasoning,
piano mover's problem, geometric modelling
computational geometry, obstacle avoidance,

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The motion planning problem is of central importance to the fields of robotics, spatial planning, and automated design. In robotics we are interested in the automatic synthesis of robot motions, given high-level specifications of tasks and geometric models of the robot and obstacles. The "Mover's" problem is to find a continuous, collision-free path for a moving object through an environment containing obstacles. We present an implemented algorithm for the "classical" formulation of the three-dimensional Movers' problem: Given an (OVER)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Block 20 cont.

arbitrary rigid polyhedral moving object "P" with three translational and three rotational degrees of freedom, find a continuous, collision-free path taking "P" from some initial configuration to a desired goal configuration.

This thesis describes the first known implementation of a complete algorithm (at a given resolution) for the full six degree of freedom Movers' problem. The algorithm transforms the six degree of freedom planning problem into a point navigation problem in a six-dimensional configuration space (called C-Space). The C-Space obstacles, which characterize the physically unachievable configurations, are directly represented by six-dimensional manifolds whose boundaries are five dimensional C-surfaces. By characterizing these surfaces and their intersections, collision-free paths may be found by the closure of three operators which (i) slide along 5-dimensional level C-surfaces parallel to C-Space obstacles; (ii) slide along 1- to 4-dimensional intersections of level C-surfaces; and (iii) jump between 6-dimensional obstacles.

Implementing the point navigation operators requires solving fundamental representational and algorithmic questions: we will derive new structural properties of the C-Space constraints and show how to construct and represent C-surfaces and their intersection manifolds. A definition and new theoretical results are presented for a six-dimensional C-Space extension of the generalized Voronoi diagram, called the C-Voronoi diagram, whose structure we relate to the C-surface intersection manifolds. The representations and algorithms we develop impact many geometric planning problems, and extend to Cartesian manipulators with six degrees of freedom.

Originator-supplied keywords include: [illegible]

Motion Planning with Six Degrees of Freedom

by

Bruce Randall Donald

This report is a revised version of chapters 1–7 of "*Local and Global Techniques for Motion Planning,*" a thesis submitted on May 10, 1984 to the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science. Chapters 8–11 may be obtained seperately as A.I. Memo 736, "*Hypothesizing Channels Through Free-Space in Solving the Findpath Problem.*"

Motion Planning with Six Degrees of Freedom

by

Bruce Randall Donald

Abstract: The motion planning problem is of central importance to the fields of robotics, spatial planning, and automated design. In robotics we are interested in the automatic synthesis of robot motions, given high-level specifications of tasks and geometric models of the robot and obstacles. The *Mover's* problem is to find a continuous, collision-free path for a moving object through an environment containing obstacles. We present an implemented algorithm for the *classical* formulation of the three-dimensional Movers' problem: Given an arbitrary rigid polyhedral moving object P with three translational and three rotational degrees of freedom, find a continuous, collision-free path taking P from some initial configuration to a desired goal configuration.

This thesis describes the first known implementation of a complete algorithm (at a given resolution) for the full six degree of freedom Movers' problem. The algorithm transforms the six degree of freedom planning problem into a point navigation problem in a six-dimensional configuration space (called *C-Space*). The *C-Space* obstacles, which characterize the physically unachievable configurations, are directly represented by six-dimensional manifolds whose boundaries are five dimensional *C-surfaces*. By characterizing these surfaces and their intersections, collision-free paths may be found by the closure of three operators which (i) slide along 5-dimensional level C-surfaces parallel to *C-Space* obstacles; (ii) slide along 1- to 4-dimensional intersections of level C-surfaces; and (iii) jump between 6-dimensional obstacles.

Implementing the point navigation operators requires solving fundamental representational and algorithmic questions: we will derive new structural properties of the *C-Space* constraints and show how to construct and represent C-surfaces and their intersection manifolds. A definition and new theoretical results are presented for a six-dimensional *C-Space* extension of the generalized Voronoi diagram, called the *C-Voronoi diagram*, whose structure we relate to the C-surface intersection manifolds. The representations and algorithms we develop impact many geometric planning problems, and extend to Cartesian manipulators with six degrees of freedom.

Acknowledgements. This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's Artificial Intelligence research is provided in part by the System Development Foundation, in part by the Office of Naval Research under Office of Naval Research contract N00014-81-K-0494, and in part by the Advanced Research Projects Agency under Office of Naval Research contracts N00014-80-C-0505 and N00014-82-K-0334.

Acknowledgments

This work was made possible by many friends, mathematicians, and scientists at the A.I. Lab. While tradition requires me to take responsibility for any remaining flaws, honesty compels me to share credit with them for whatever insight and clarity this thesis manifests.

I am deeply indebted to my supervisor, Tomás Lozano-Pérez, for his guidance, support, and encouragement. Many of the key ideas in this thesis arose in conversations with Tomás, and this work would have been impossible without his help.

Thanks to Patrick Winston for providing generous support and the unique environment of the A.I. Lab. In particular, thanks for the private VAX and Lisp Machine I required for the implementation.

Michael Erdmann, John Canny, and Steve Buckley showed me shorter proofs and better friendship than I probably deserved. They spent many hours with me at whiteboards discussing this research. Mike was always willing to talk about math, and spent the weekend after his oral exam carefully reading a draft of this thesis and making many insightful comments and suggestions.

Thanks to Mike Brady, Rod Brooks, and Eric Grimson for reading drafts and for much encouragement. Through their comments and infectious enthusiasm, the presentation of this report was much improved. Rod shared his code and gave help with the implementation of the channels system, too.

Thanks to all the robotics and vision people, especially Philippe Brou, Rich Doyle, Ellen Hildreth, Gideon Sahar, Lori Sorayama, and Demetri Terzopoulos for discussions and help at various times. Philippe's Dover program allowed me to send lisp machine graphics to our laser printer.

I would like to thank the Macsyma Consortium for the ability to manipulate complicated equations. I am grateful to George J. Carrette for spending many hours helping to bring up Macsyma under NIL on our VAX, and for advice on how to optimize the algebra system under NIL for the Lisp Machine.

Thanks to Robin for diversion and support. Finally, thanks to my parents for incalculable help over many years.

Table of Contents

1
Geometric Planning Problems

Introduction and Statement of the Problem

The motion planning problem is of central importance to the fields of robotics, spatial planning, and automated design. In robotics we are interested in the automatic synthesis of robot motions, given high-level specifications of tasks and geometric models of the robot and obstacles. The problem is to find a continuous, collision-free path for a moving object through an environment containing obstacles; hence it has also been called the *Find-Path* or *Piano Movers'* problem. In its most general formulation the object can have an arbitrary number of hinges and joints, and in some cases coordinated motion planning for multiple objects has been considered. We will confine ourselves to the *classical* formulation of the Movers' problem: Given an arbitrary rigid polyhedral moving object P, find a continuous, collision-free path taking P from some initial configuration to a desired goal configuration. We are particularly interested in the 3-dimensional Movers' problem, for an object with 3 translational and 3 rotational degrees of freedom. This thesis describes the first known implementation of a complete algorithm (at a given resolution) for the full 6 degree of freedom Movers' problem.

1.1. What are Geometric Planning Problems?

Our work has impact on a class of *geometric planning problems*. In robotics we are typically interested in motion planning for a mobile robot or manipluator.

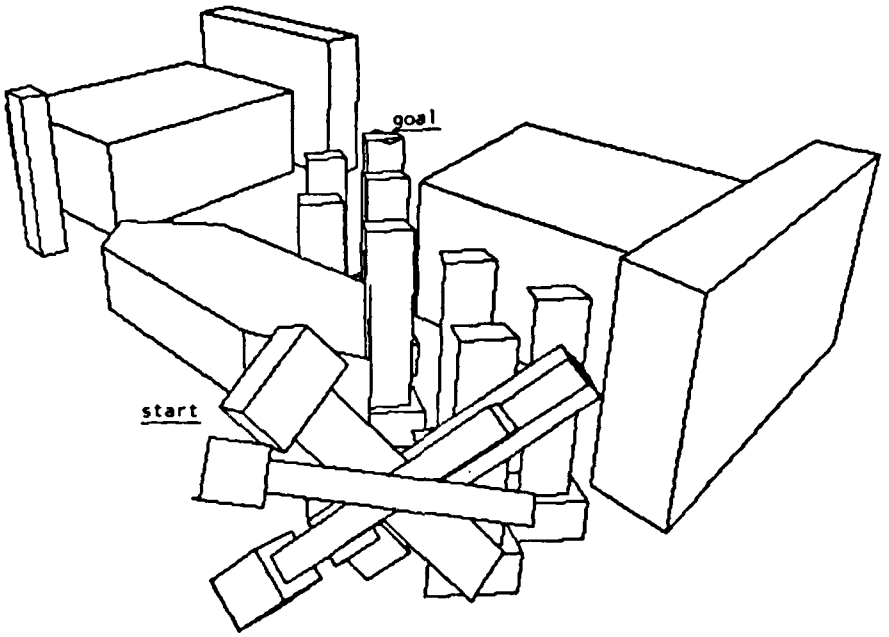


Figure 1.1. An example of a solution path for the classical Mover's problem with six degrees of freedom. This illustration is a "time-lapse" picture of a path found by our planner for a hammer-shaped object. In all our examples, the workspace is bounded by a box (which is not shown). This solution path requires use of all three rotational degrees of freedom.

In Computer-Aided Design (CAD), the problem of automated structural design for n structural members is also an instance of the most general form of the Mover's problem. The problem of determining whether an object can be assembled as designed, and of generating an assembly plan if it can, is also in this class. Examples of geometric planning problems include:

(i) The *Find-Path* or *Movers'* problem is to find continuous, collision-free path for one or more moving objects in the presence of obstacles. Find-path problems fall into two broad categories: single-body and multiple-body motion planning problems. In the *classical* Movers' problem, a single rigid polyhedral object must be moved through a workspace containing polyhedral obstacles. For the *linked* or *hinged body* Movers' problem, a set of moving objects connected *via* joints or linkages must be moved. An industrial robot arm is a typical example of such an

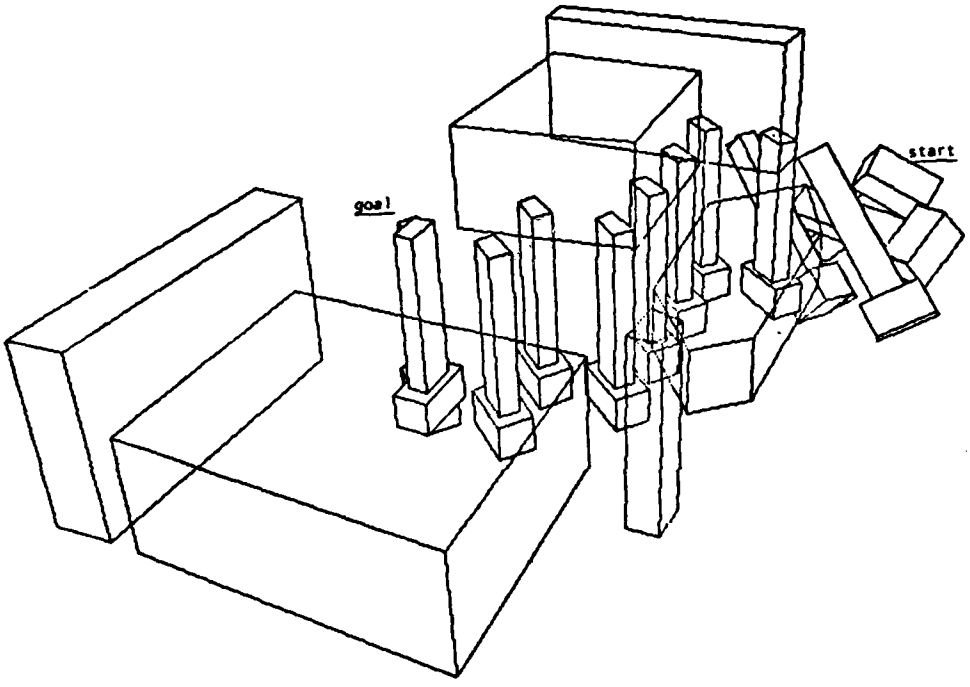


Figure 1.2. A different view of the solution path for the hammer example, with the obstacles "transparent" to allow us to view the rotations better.

object. In the *coordinated* planning problem, a number of independent (i.e., not necessarily linked) objects must be moved. An algorithm for multiple body motion planning must ensure that the moving objects collide neither with the walls nor with each other.

(ii) The *find-space* problem is to find a collision-free placement for one or more objects in a field of obstacles. By analogy with the find-path problem, we can speak of the classical, linked-body, and coordinated find-space problems. In computer-aided design and automated design, the find-space problem is typically subject to additional geometric constraints. Lozano-Pérez (1983) grouped find-path and find-space algorithms together as the *spatial planning* problems.

(iii) The *fine-motion* problem entails motion-planning along obstacle surfaces, typically while maintaining some applied force. Collision-free paths and placements avoid obstacles: however, for many tasks in robotics and in automated design, it

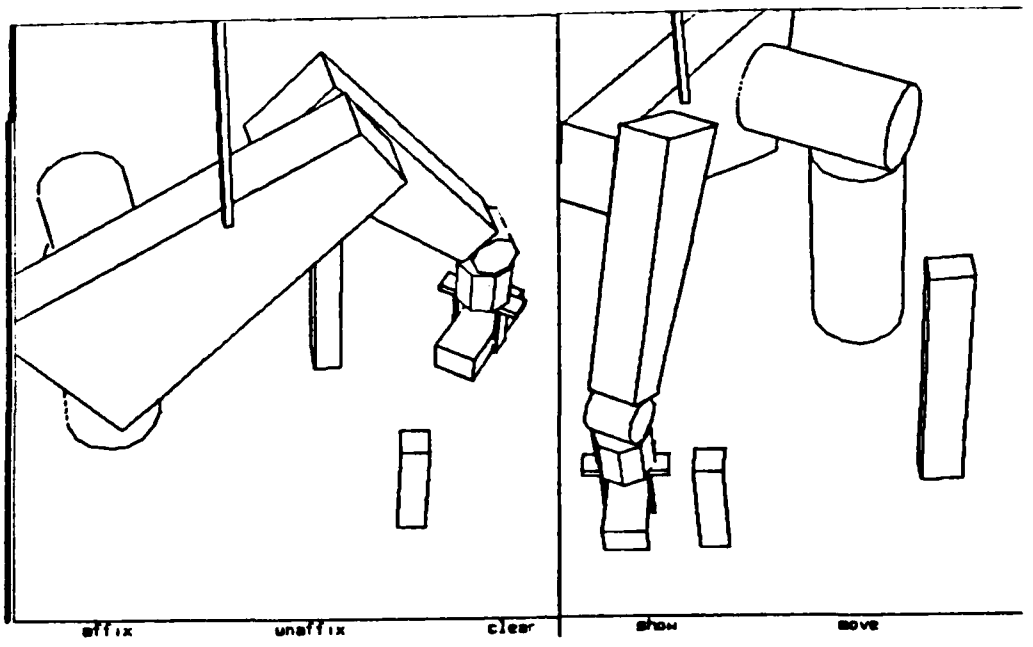


Figure 1.3. Application Example: Planning for an industrial robot arm with six degrees of freedom is an example of the linked-body movers' problem. (Figure courtesy of Rodney Brooks).

is necessary to plan motions and placements in contact with obstacle surfaces. For example, consider the tasks of welding, insertion, and assembly in robotics. These tasks require *compliant motions*, entailing consideration of additional physical constraints such as friction, kinematics, and force control. However, the compliant motion planning problem has a strong geometric flavor and its solution requires the tools of spatial planning (see Mason (1981), Erdmann (1984)).

(iv) Recently, researchers have begun to consider motion planning with *uncertainty* (Mason (1981), Brooks (1982), Lozano-Pérez, Mason, and Taylor (1983), Erdmann (1984)). Broadly speaking, uncertainty may arise from inaccuracy in object models, sensors, or control. Motion planning with uncertainty also presumes algorithms and representations from spatial planning.

As we can see, all geometric planning problems contain components of the spatial planning problem, especially if the underlying geometries are the same. In

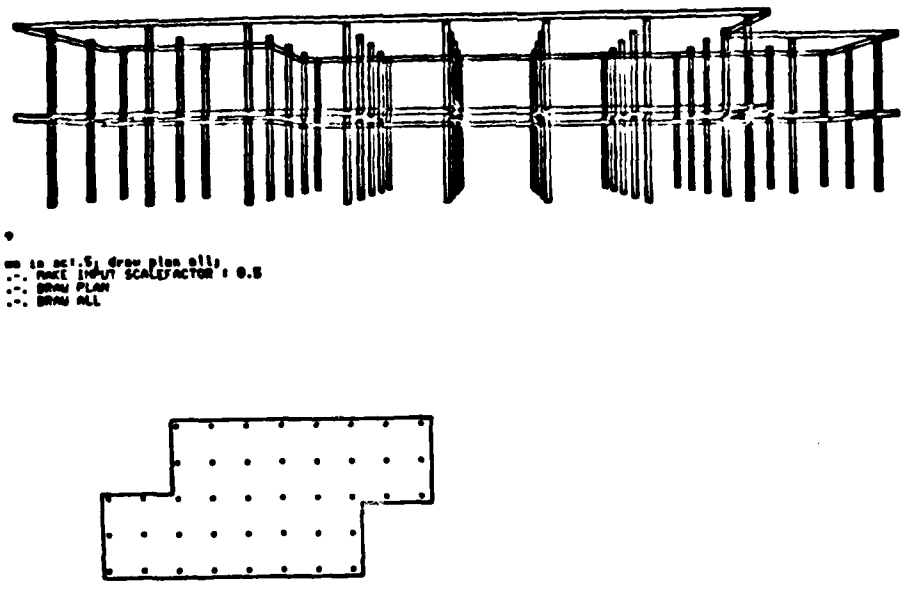


Figure 1.4. Example from computer-aided design: Automatically generated flat-plate structure from Donald (1983b). How can we generate structural patterns subject to the constraints of the building envelope and mechanical core?

particular, for high-dimensional configuration spaces, the theoretical analyses of Mason (1981), Lozano-Pérez, Mason, and Taylor (1983), and Erdmann (1984) all presume geometric results which are derived in this thesis.

This work impacts all geometric planning problems. To illustrate the theoretical results, we address one particular problem, namely the classical Movers' problem with six degrees of freedom. Our algorithms immediately generalize to applications involving gross-motion and fine-motion planning for Cartesian manipulators with six degrees of freedom.

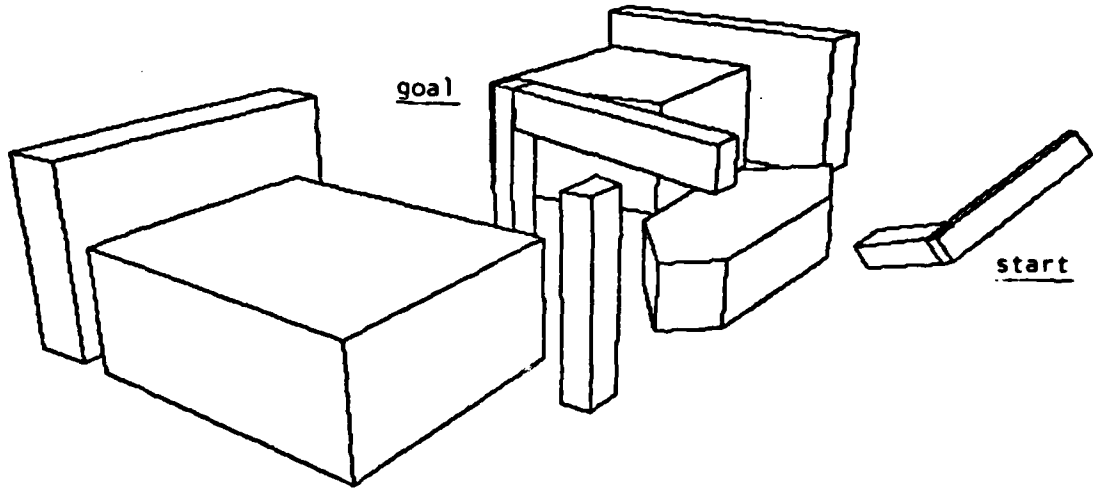


Figure 1.5. A find-path problem for an L-shaped object. The L-shaped object is shown amidst obstacles in the start and goal configurations.

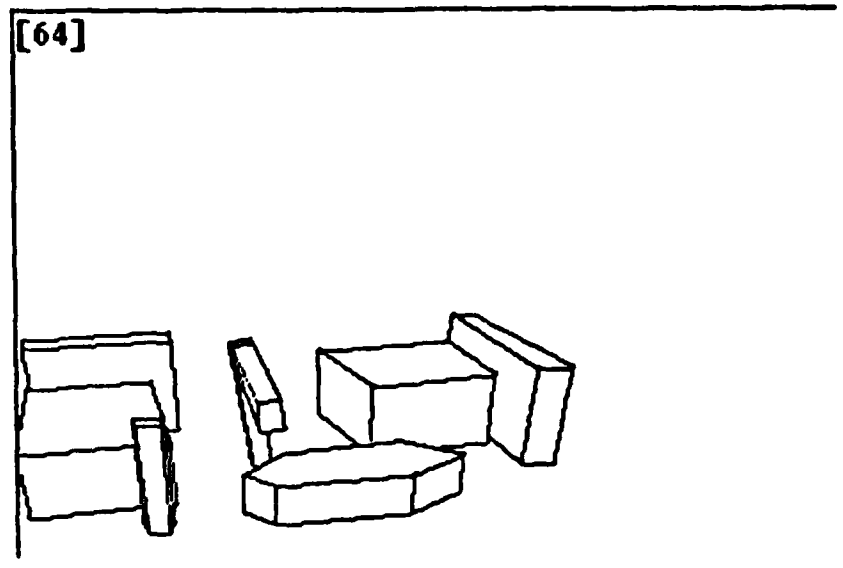


Figure 1.13. Solution Path 1, frame 64 (final configuration).

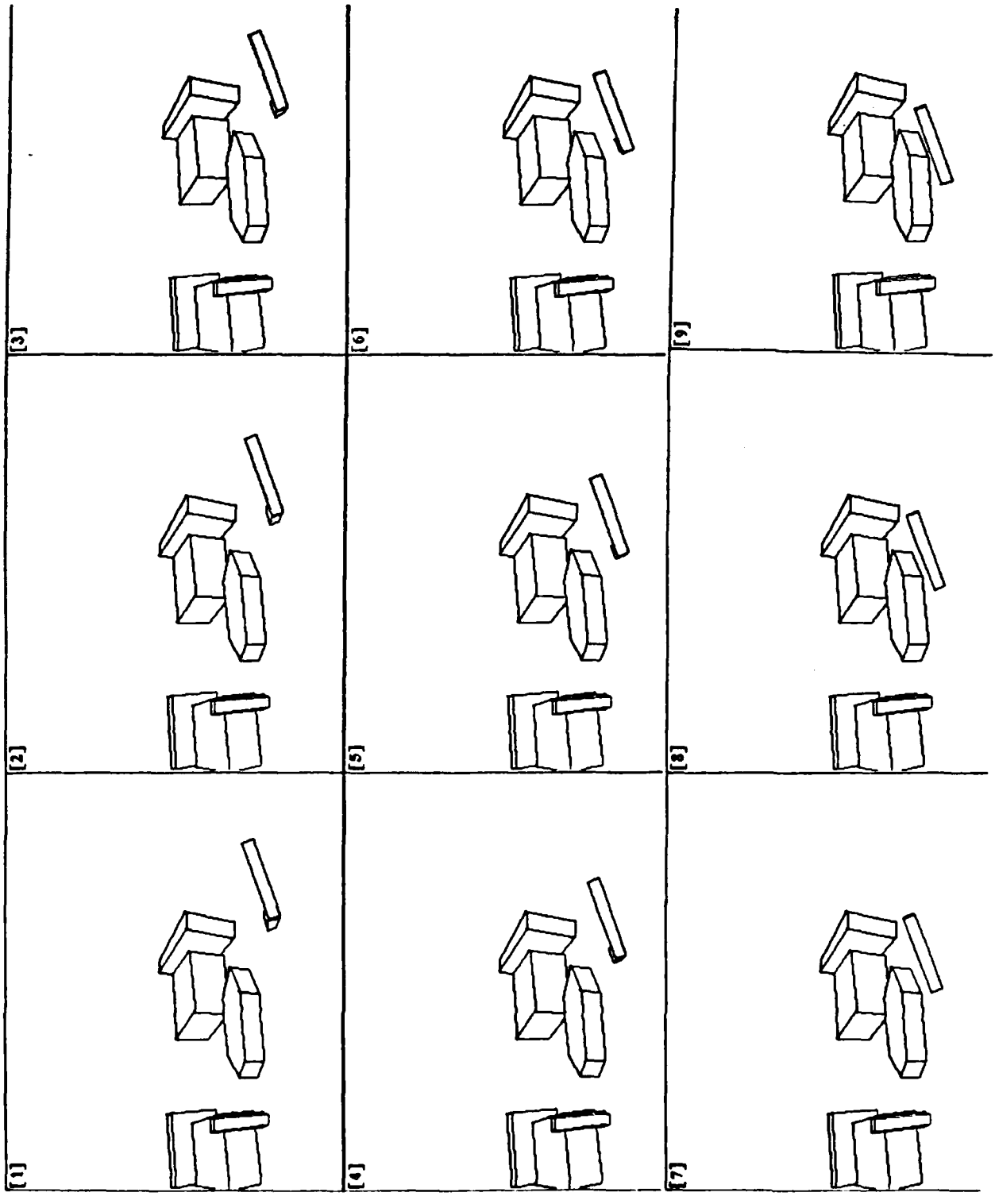


Figure 1.6. Solution Path I, frames 1 9: A difficult solution path for the L-shaped object.

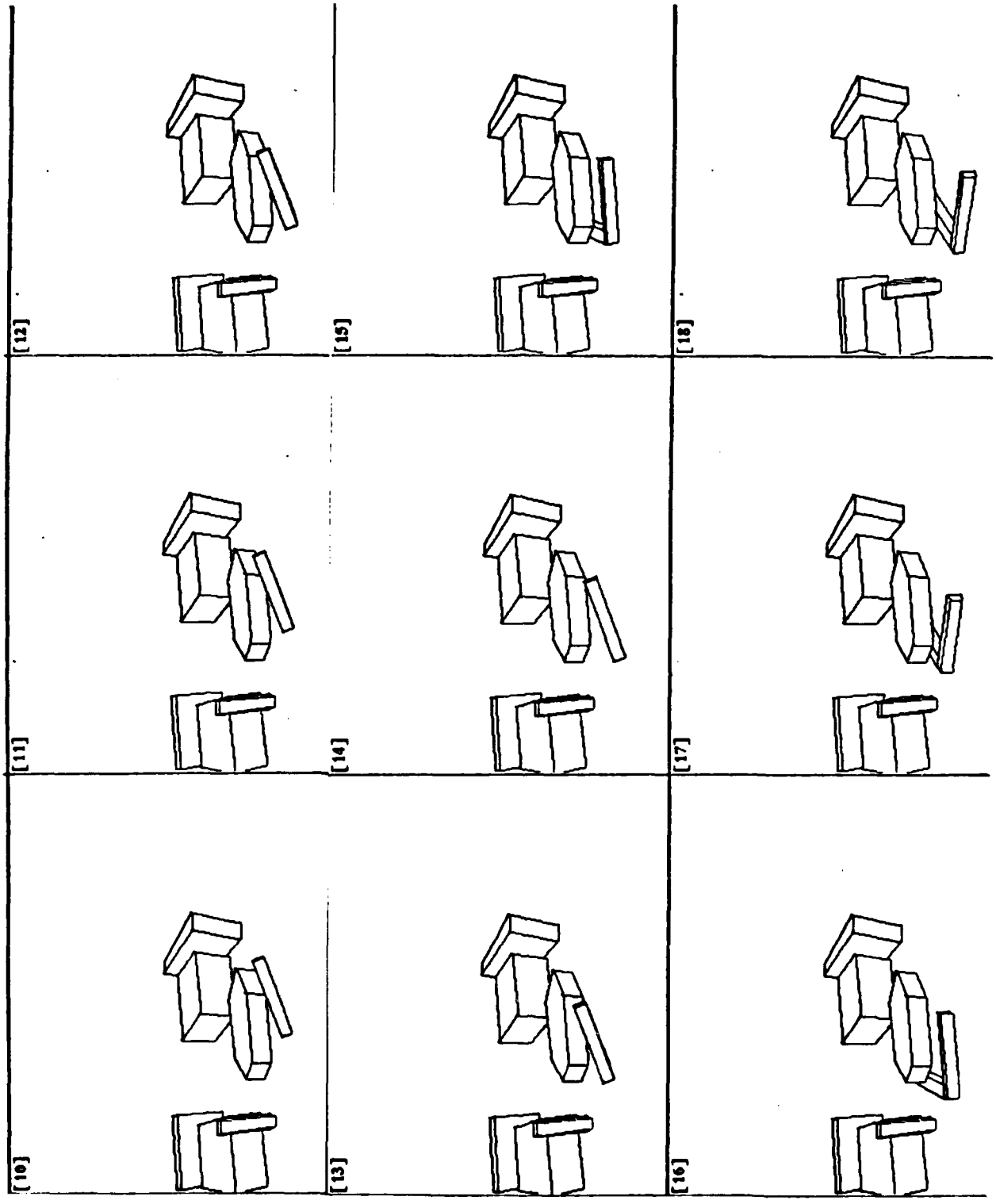


Figure 1.7. Solution Path I, frames 10-18

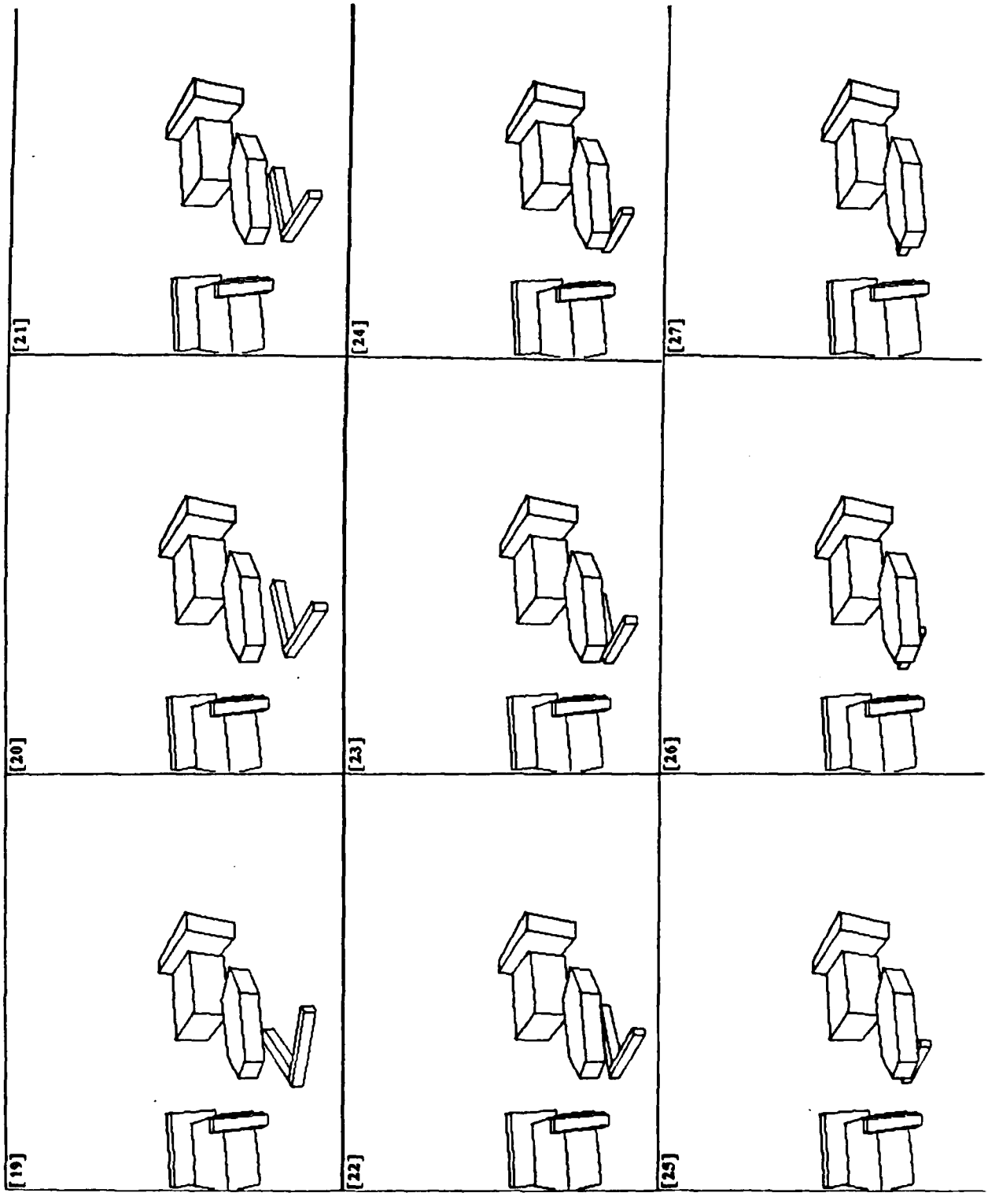


Figure 1.8. Solution Path I, frames 19-27

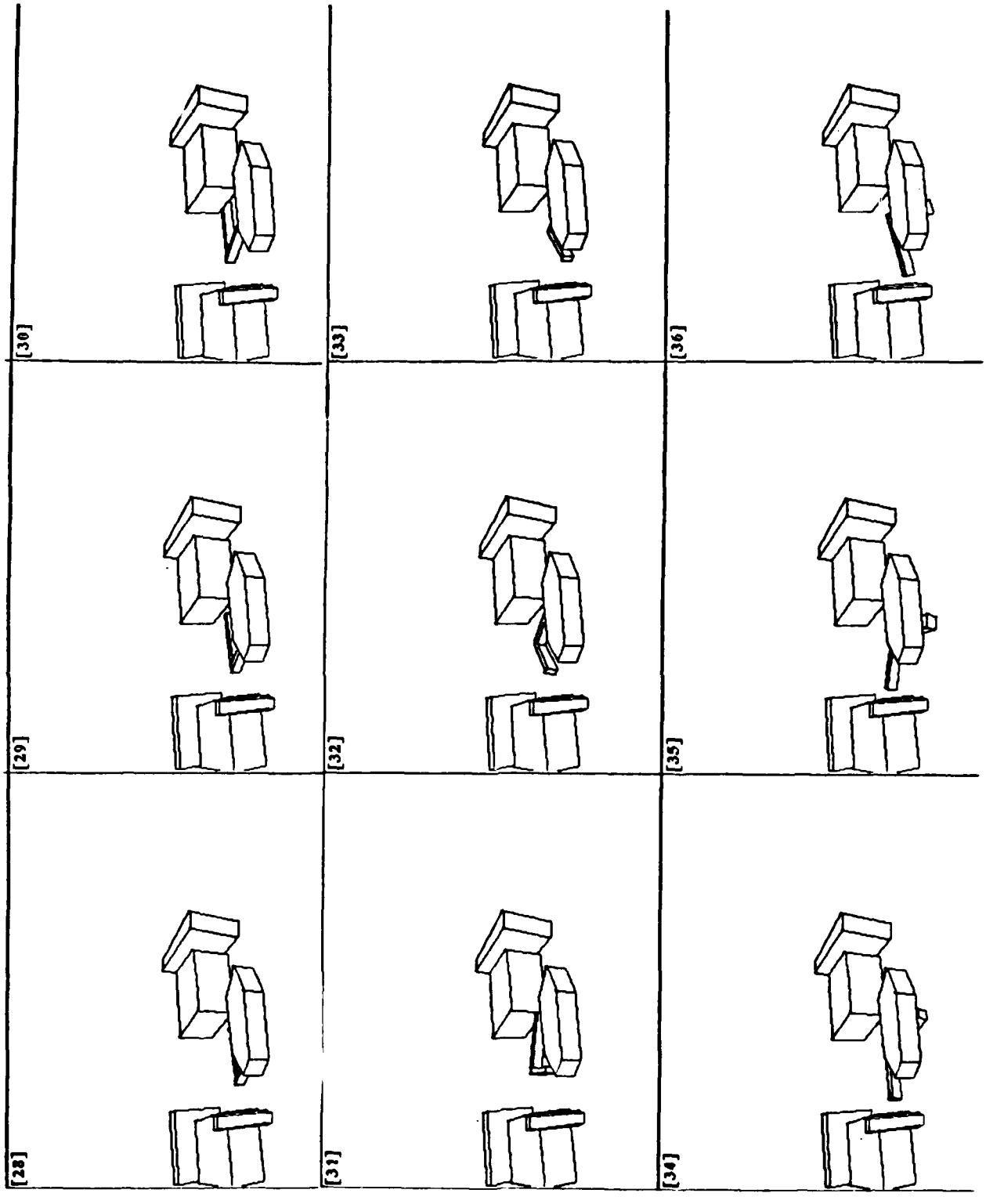


Figure 1.9. Solution Path I, frames 28 36

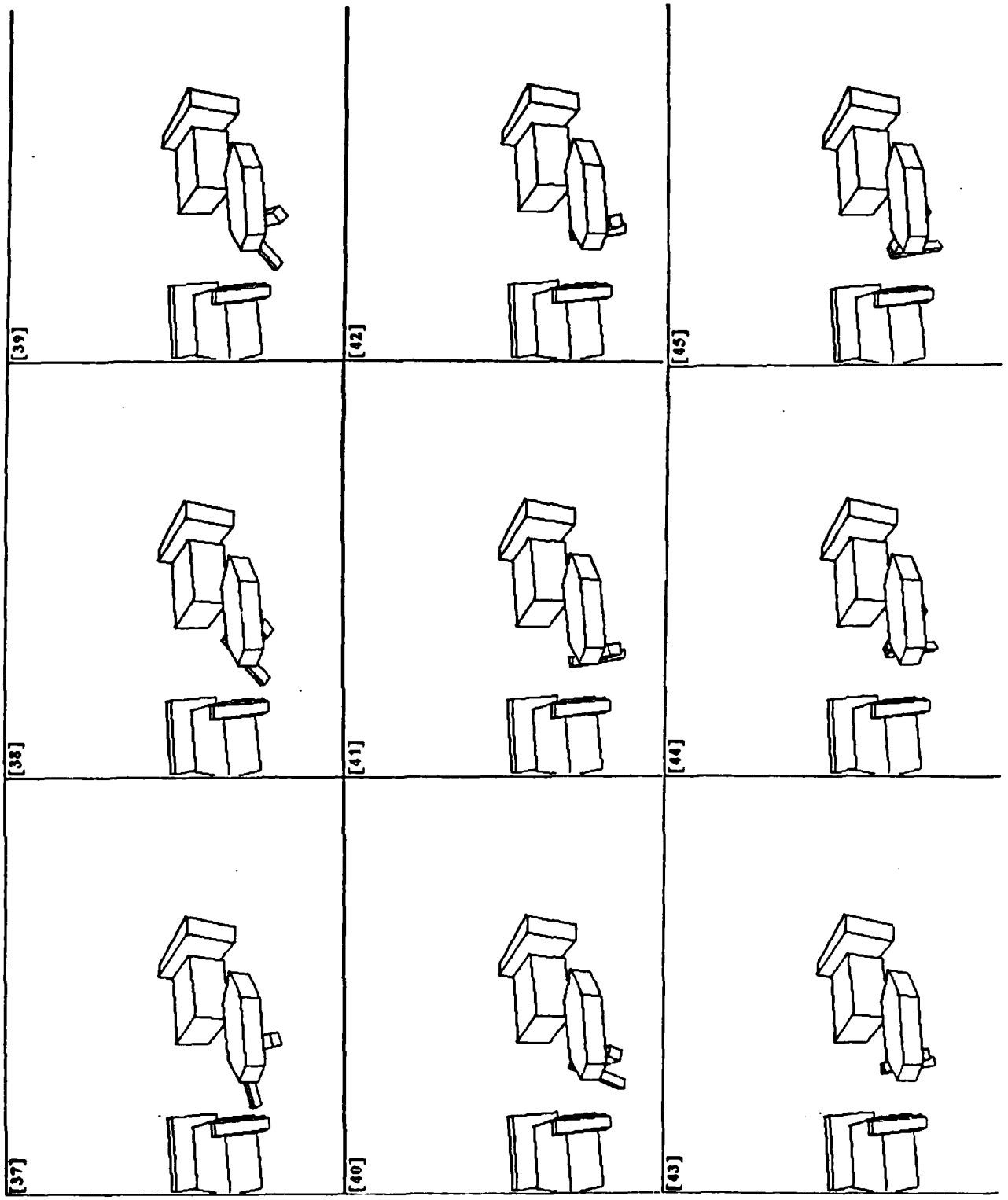


Figure 1.10. Solution Path I, frames 37-45

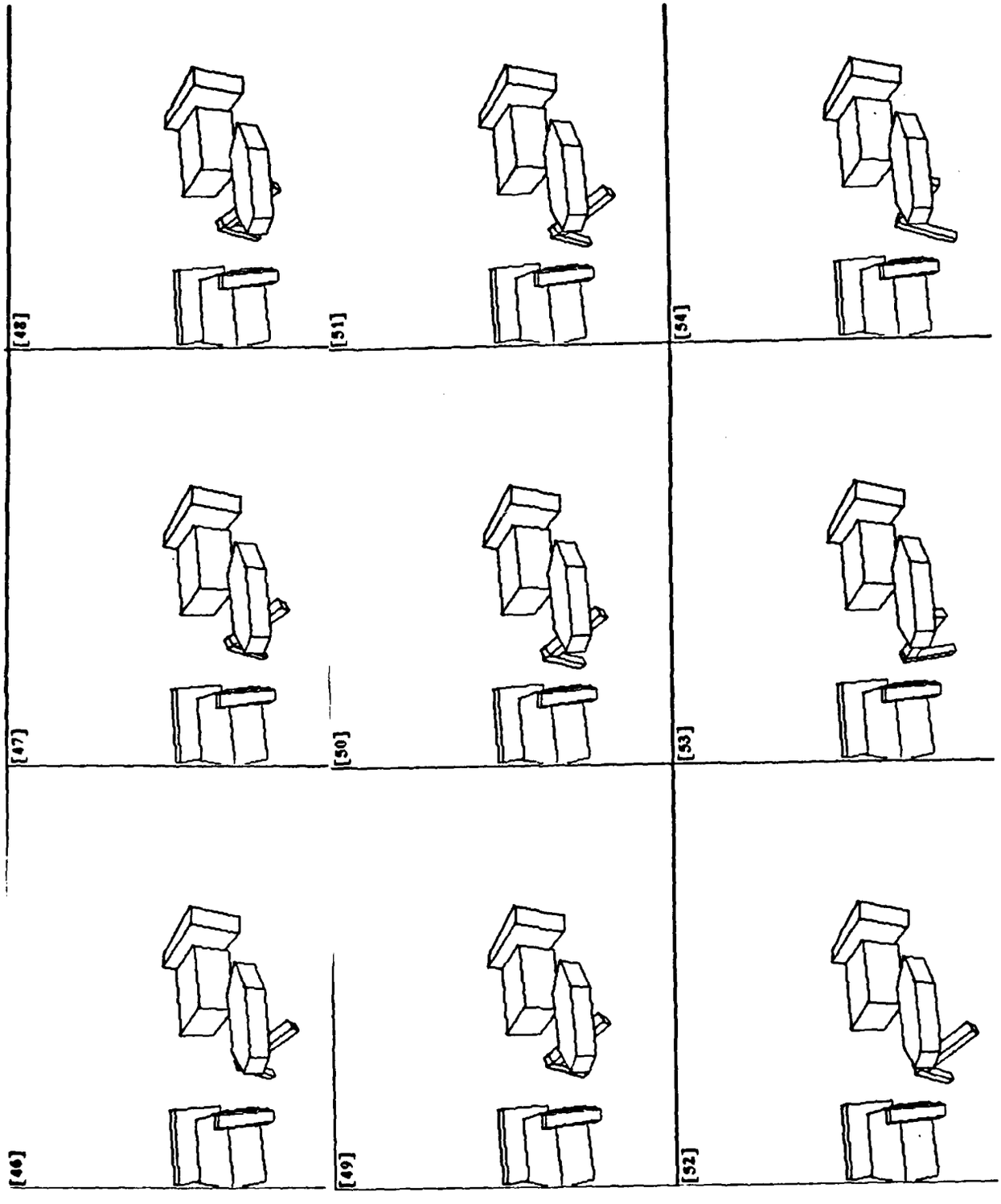


Figure 1.11. Solution Path I, frames 46 54

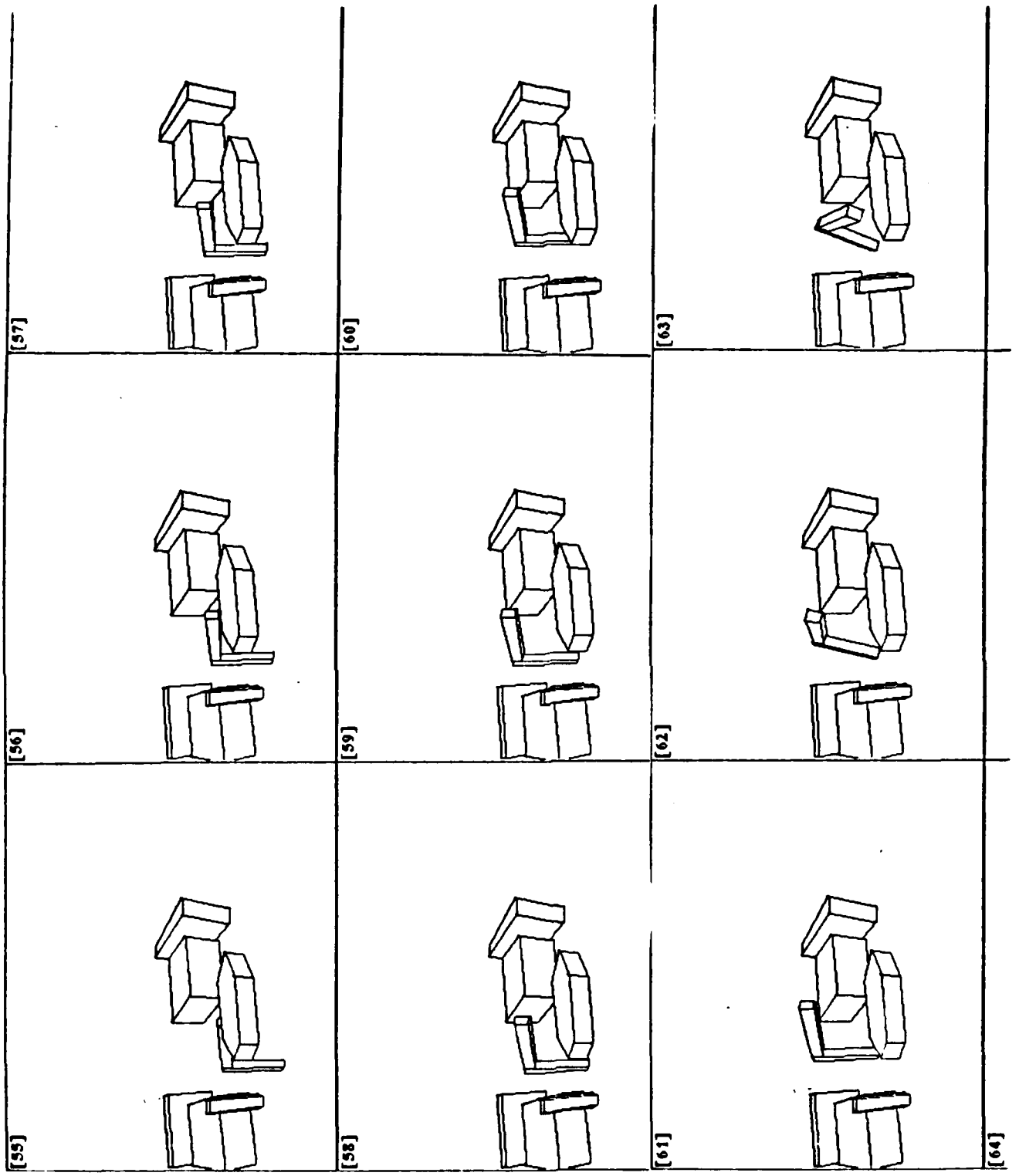


Figure 1.12. Solution Path I, frames 55 63

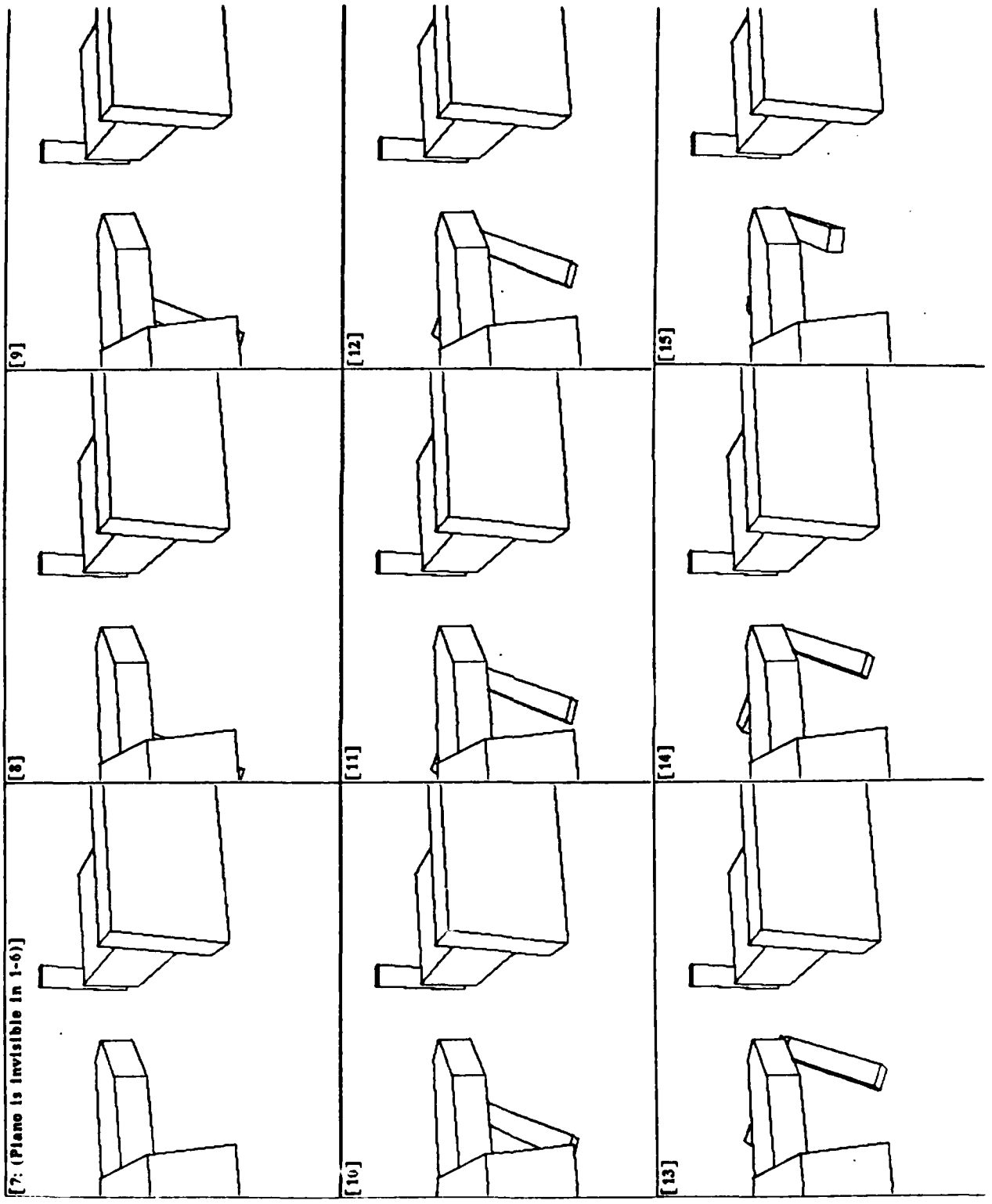


Figure 1.14. A different view of the same solution path, showing how the L-shaped object must rotate to attain the final position. The first six frames are not shown, since the moving is not visible from this perspective. Solution Path 1 (view 2); frames 7 15

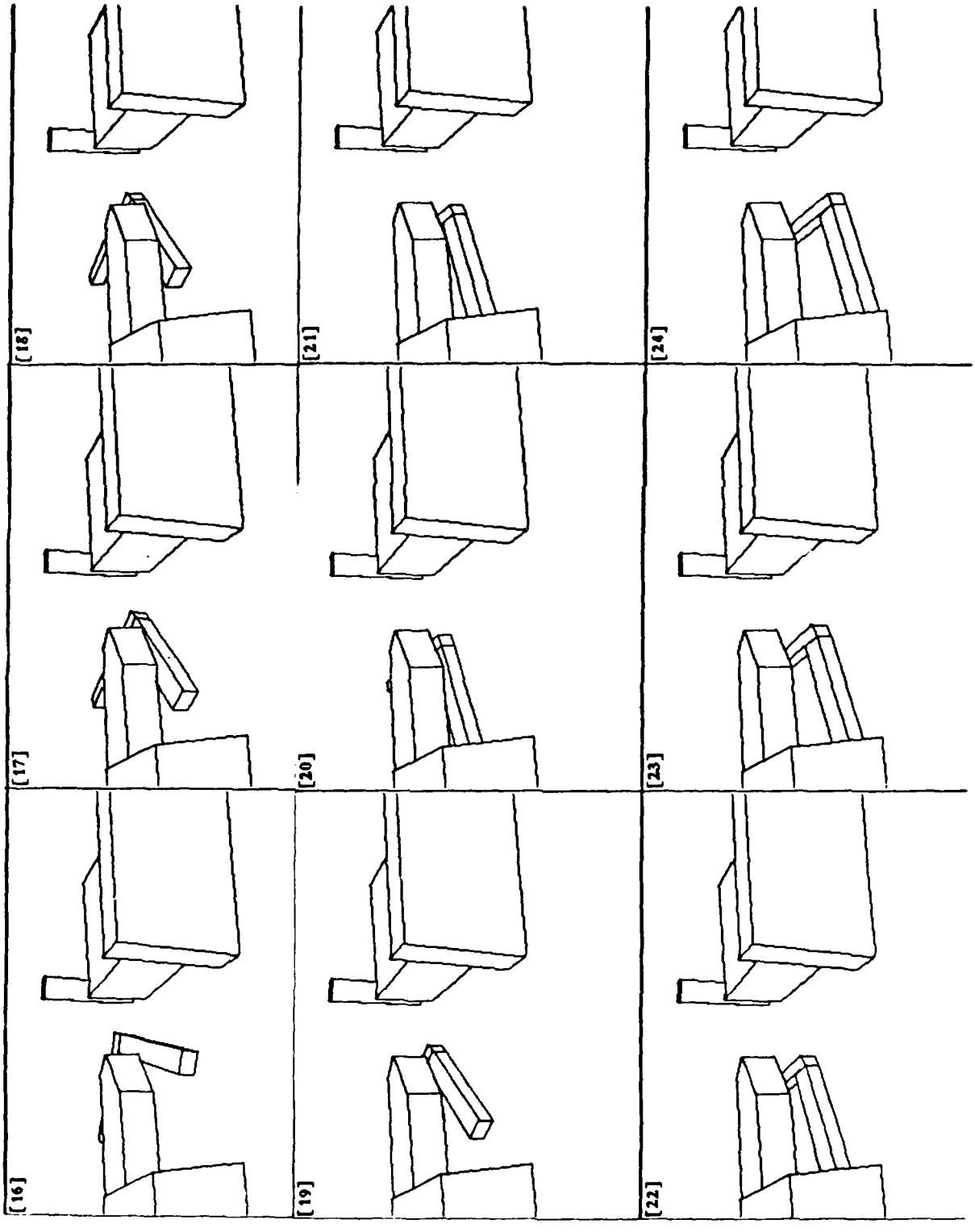


Figure 1.15. Solution Path I (view 2); frames 16–24

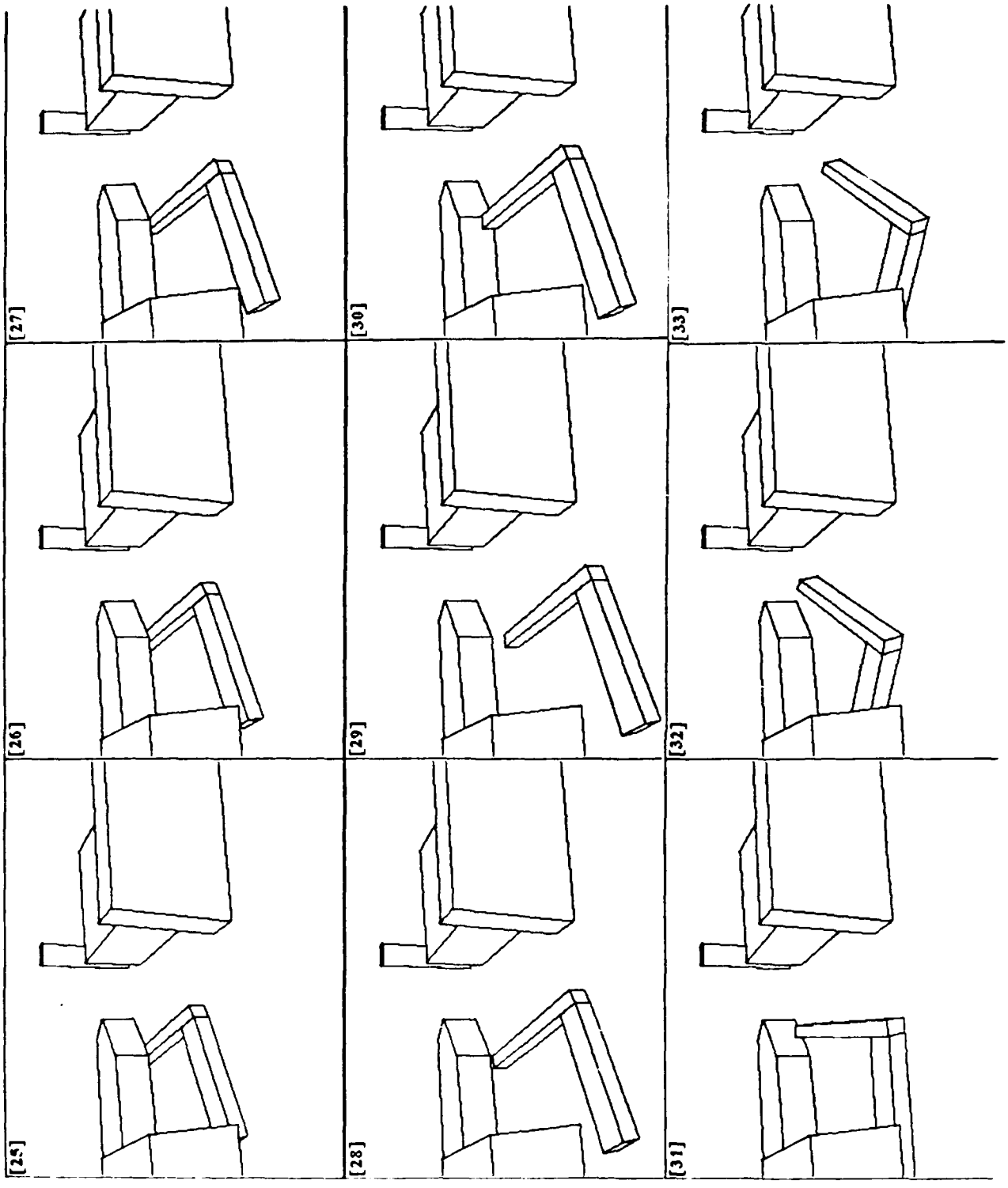


Figure 1.16. Solution Path I (view 2); frames 25 33

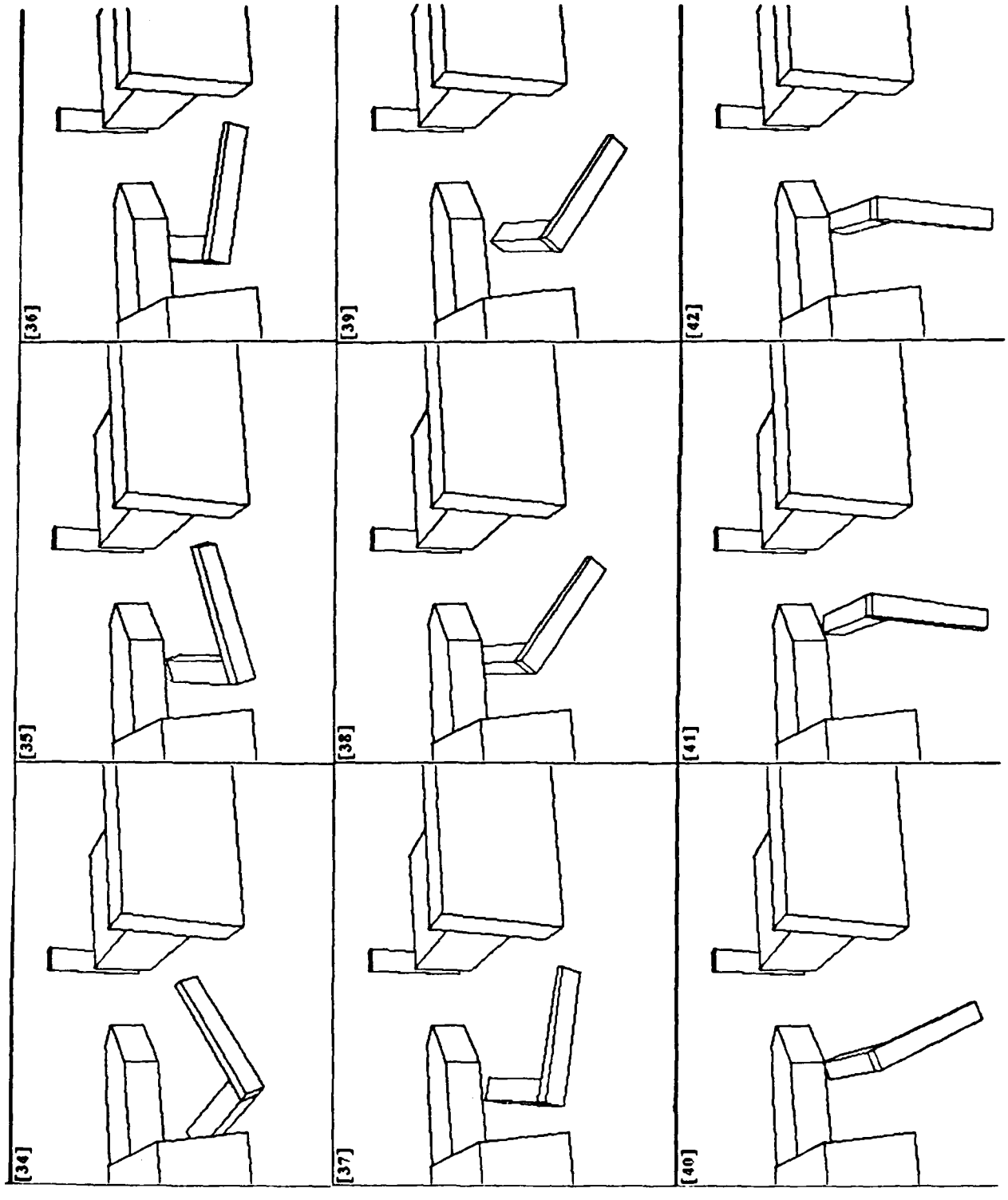


Figure 1.17. Solution Path I (view 2); frames 34-42

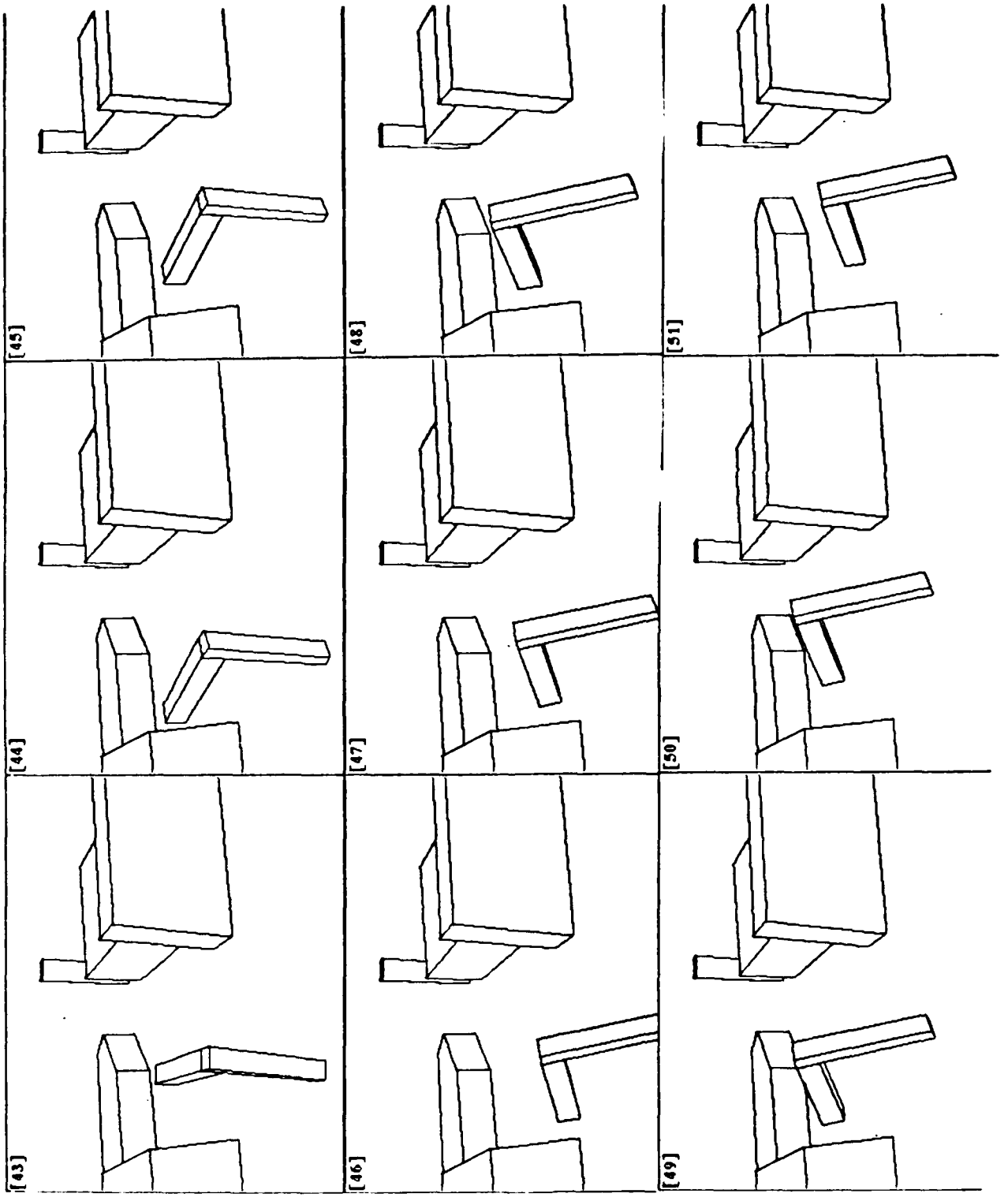


Figure 1.18. Solution Path 1 (view 2); frames 43 51

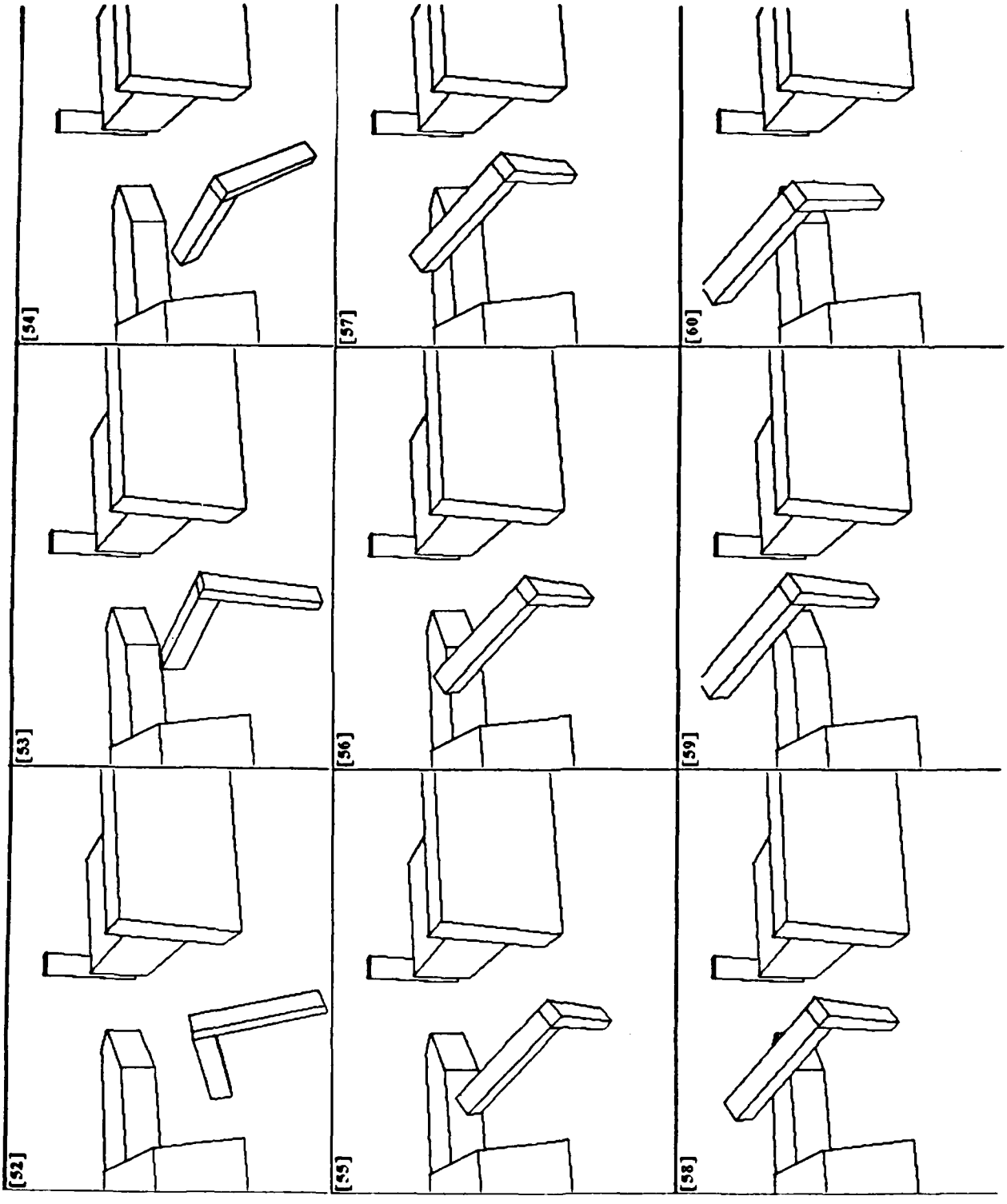


Figure 1.19. Solution Path I (view 2); frames 52 60

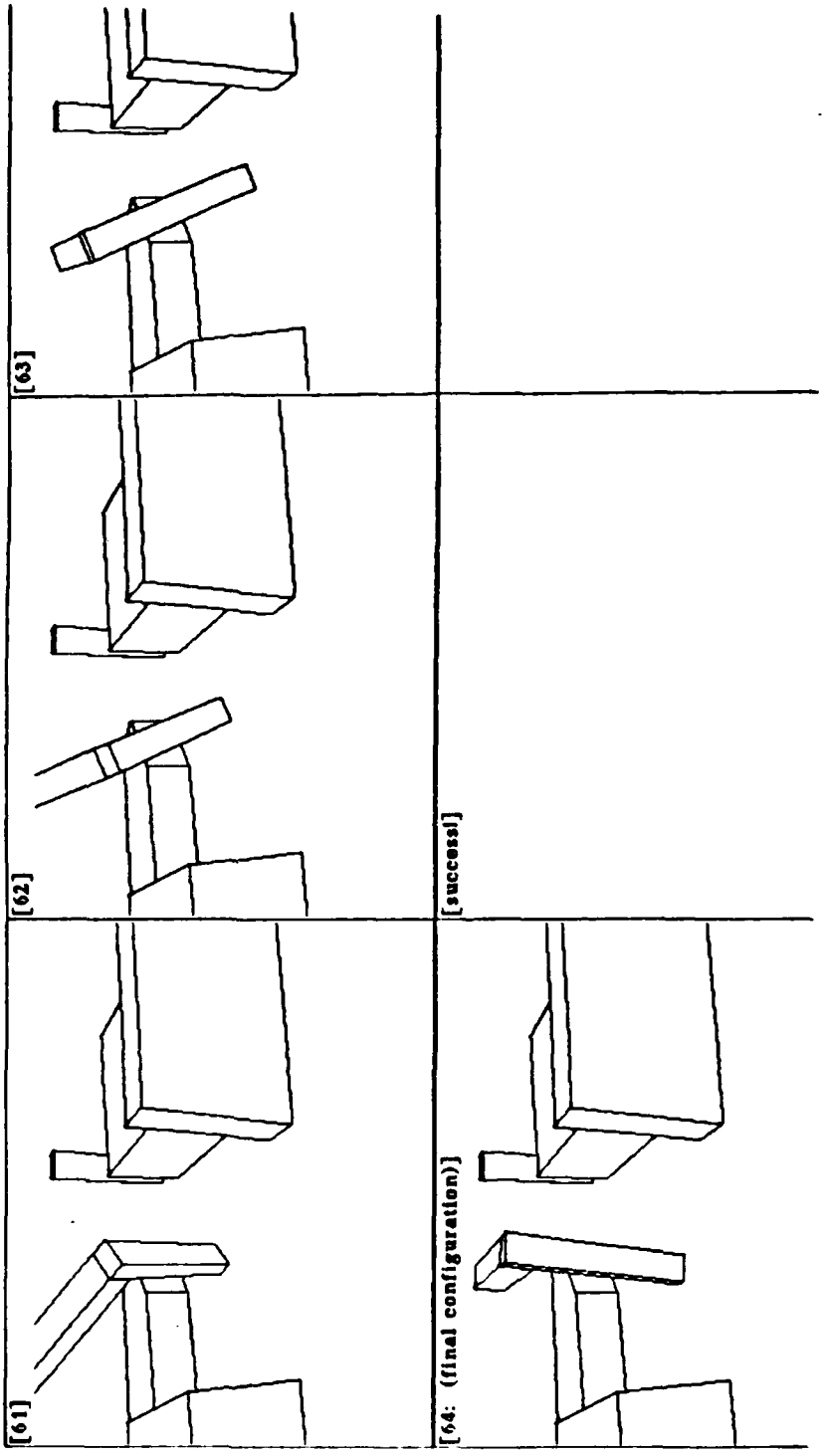


Figure 1.20. Solution Path I (view 2); frames 61-64

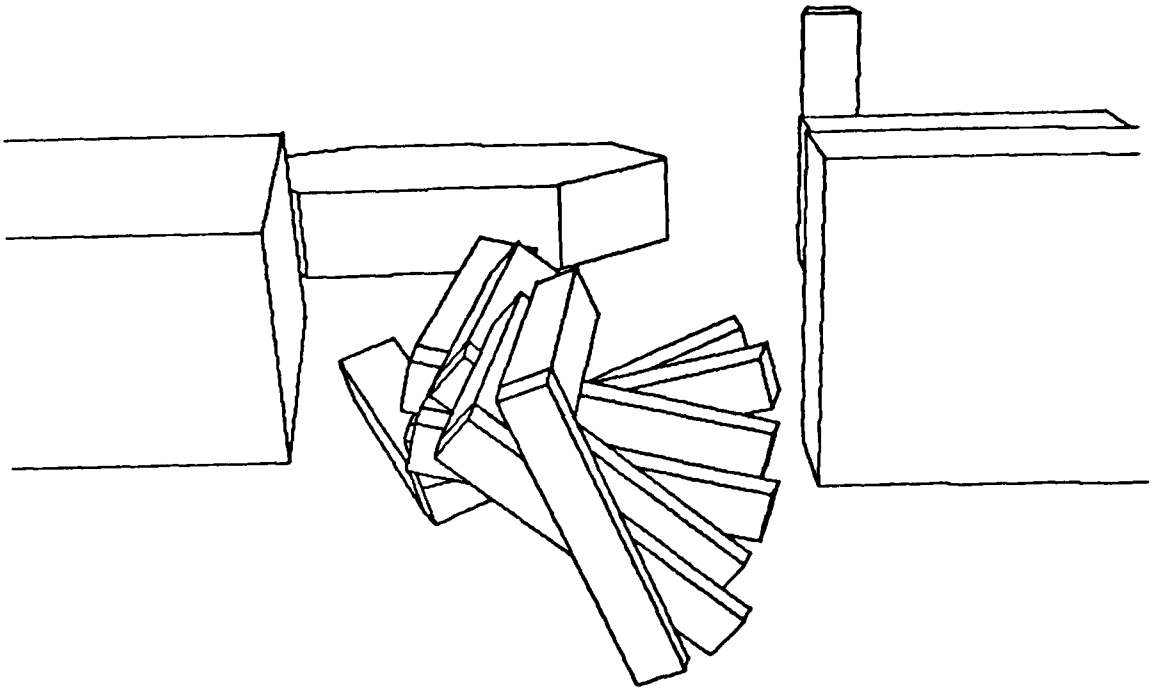


Figure 1.21. A detail of the path for the L-Shaped example. The detail is in "time-lapse" format, and shows a complex double rotation near the goal configuration.

Examples of "classical" find-path problems solved by our planner may be found throughout this chapter, and also at the end of chapter 2 (section 2.4, "*Examples of the Local Experts in Use*"). See fig. 1.5–21, 1.22–28, and 2.7–21. In general, geometric planning problems with more than three degrees of freedom have proven extremely difficult to solve. We believe that in part, this difficulty has been due to the unresolved issues in the mathematics of spatial planning. By solving these problems for the six degree of freedom case, and illustrating the results for the find-path problem (which holds considerable intrinsic interest), we hope to provide a geometric foundation which will make all geometric planning problems feasible.

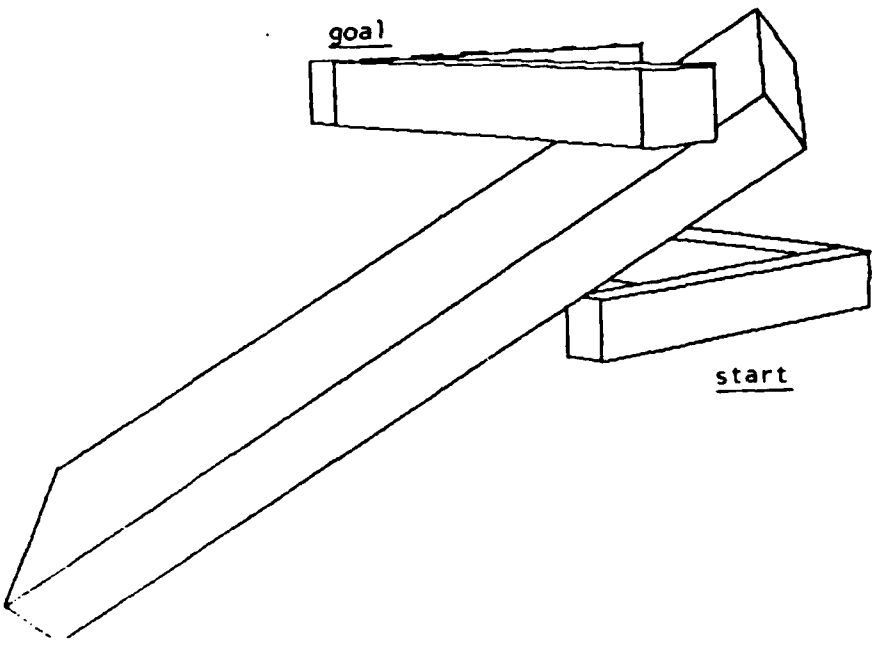


Figure 1.22. (3 Views of the "Puzzle Problem"). In this find-path problem, the L-shaped object must be moved around the diagonally-placed obstacle. Here the L-shaped object is shown in the initial and goal configurations.

1.2. A Simple Example: How to Find a Path for a Point Amidst 3-D Polyhedral Obstacles

We will begin by discussing an algorithm for navigating a single point amidst polyhedral obstacles in three-dimension Euclidean space $\Re^3$. We then review the configuration space transformation of Lozano-Pérez (1983), which transforms the problem of reasoning about the motion of a polyhedral object to reasoning about a single point in configuration space. If the configuration space is isomorphic to $\Re^3$, then the point navigation algorithm can be applied directly to find collision free paths. In this thesis we will generalize the point navigation algorithm to work in the configuration space for a three dimensional polyhedral object with six degrees of freedom.

The six degree of freedom planner is based on the following analogy: suppose

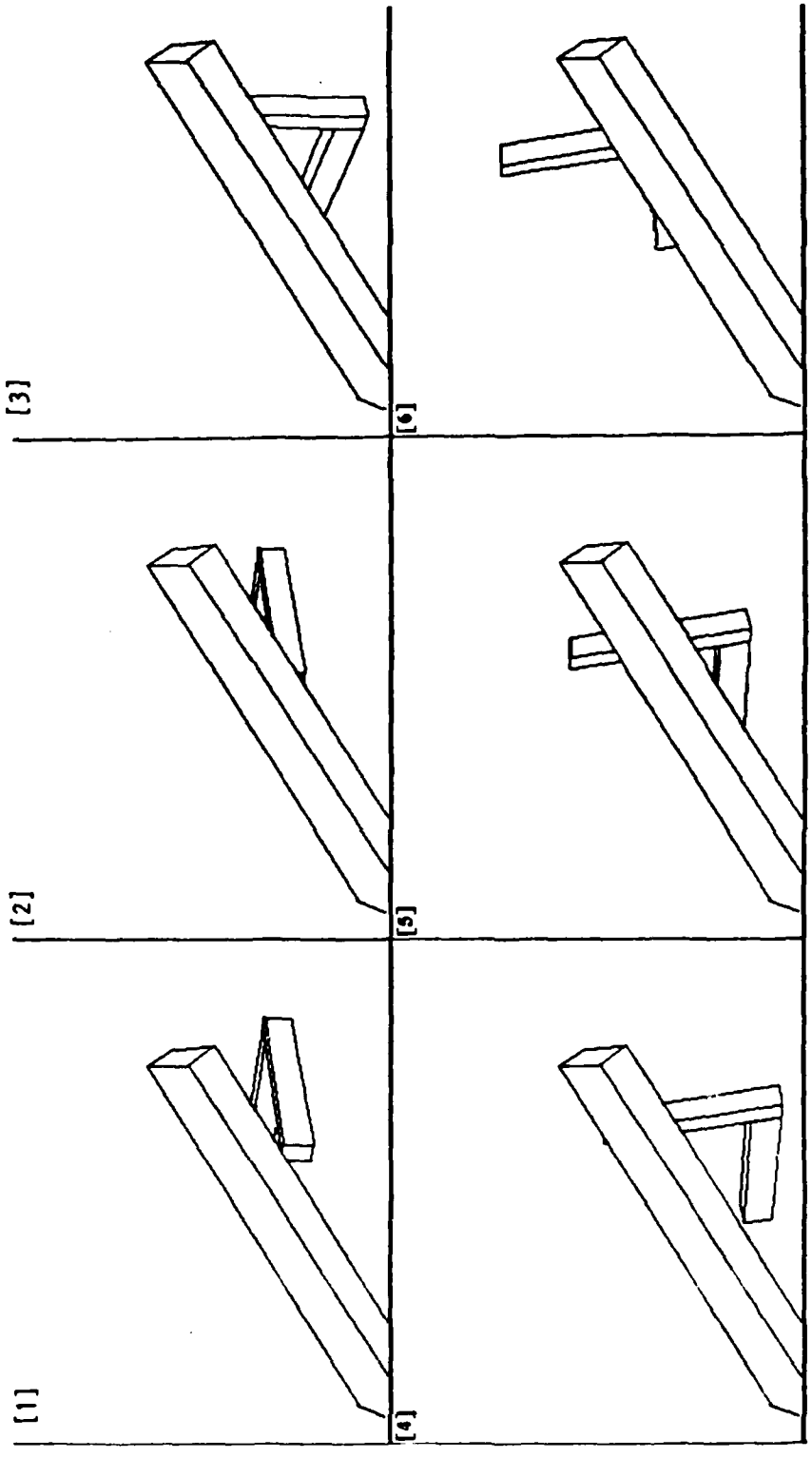


Figure 1.23. Puzzle Problem, frames (1-6), view 1.

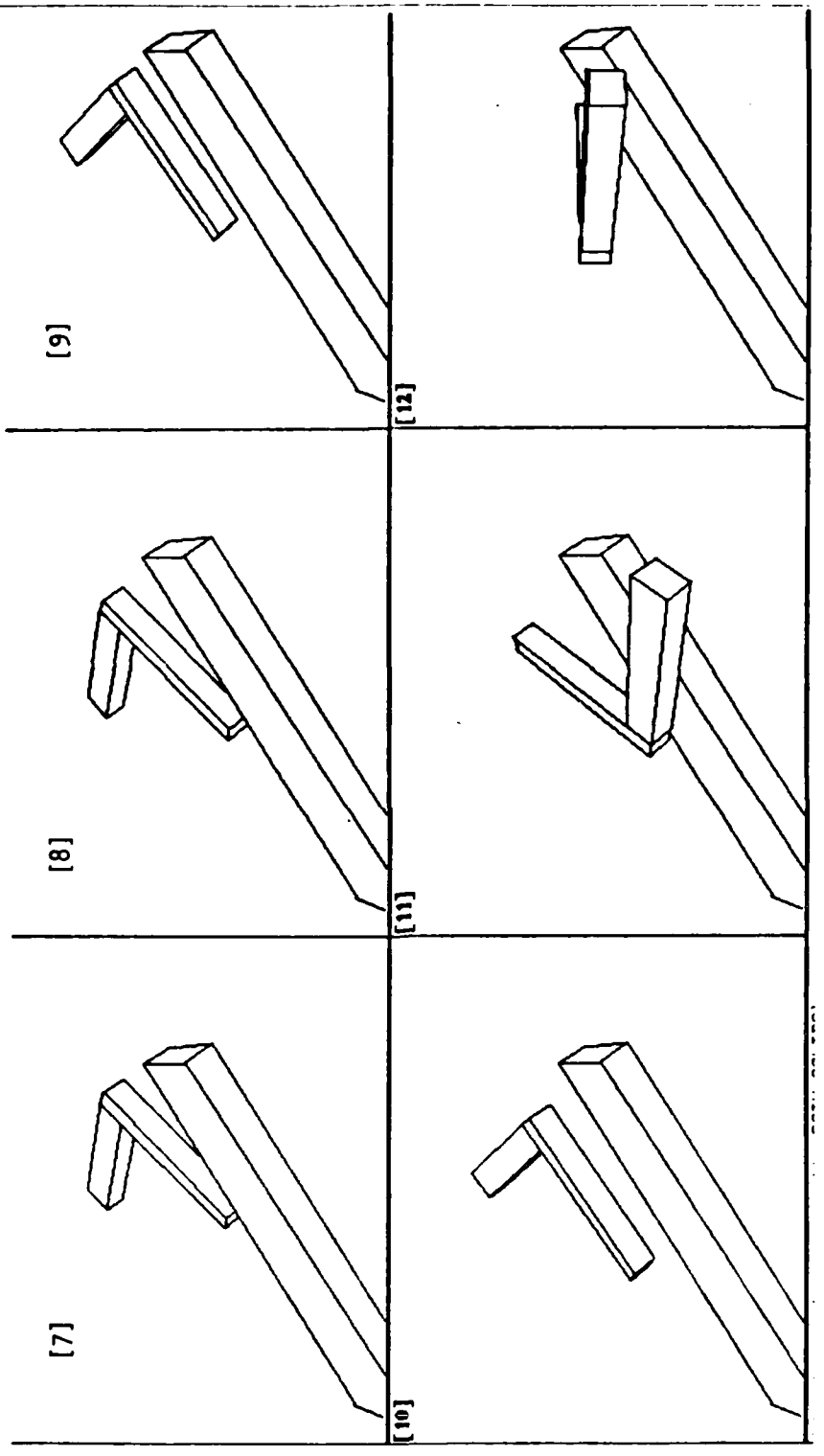


Figure 1.24. Puzzle-Problem, frames (7-12), view 1

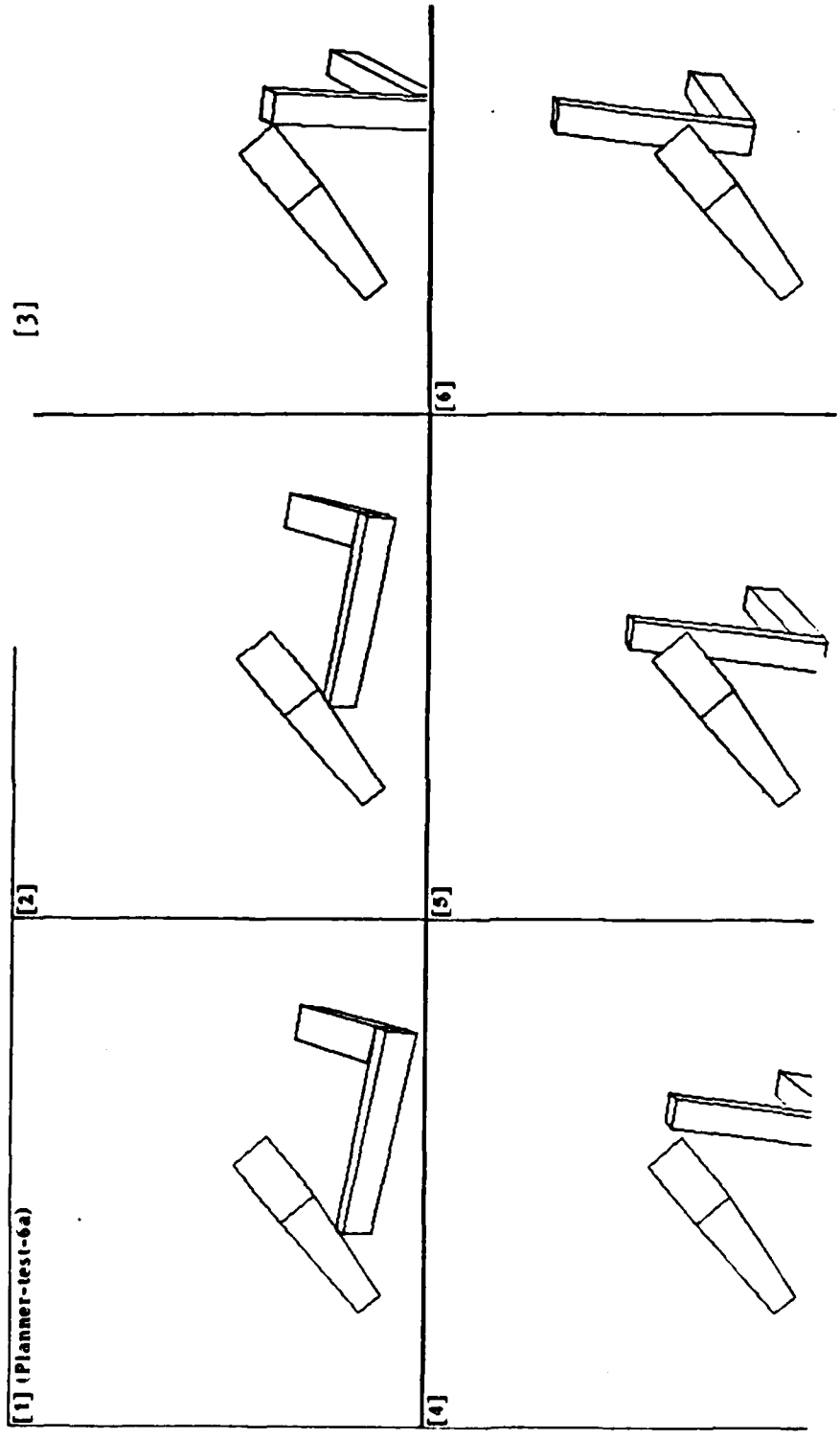


Figure 1.25. Puzzle-Problem, frames (1-6), view 2

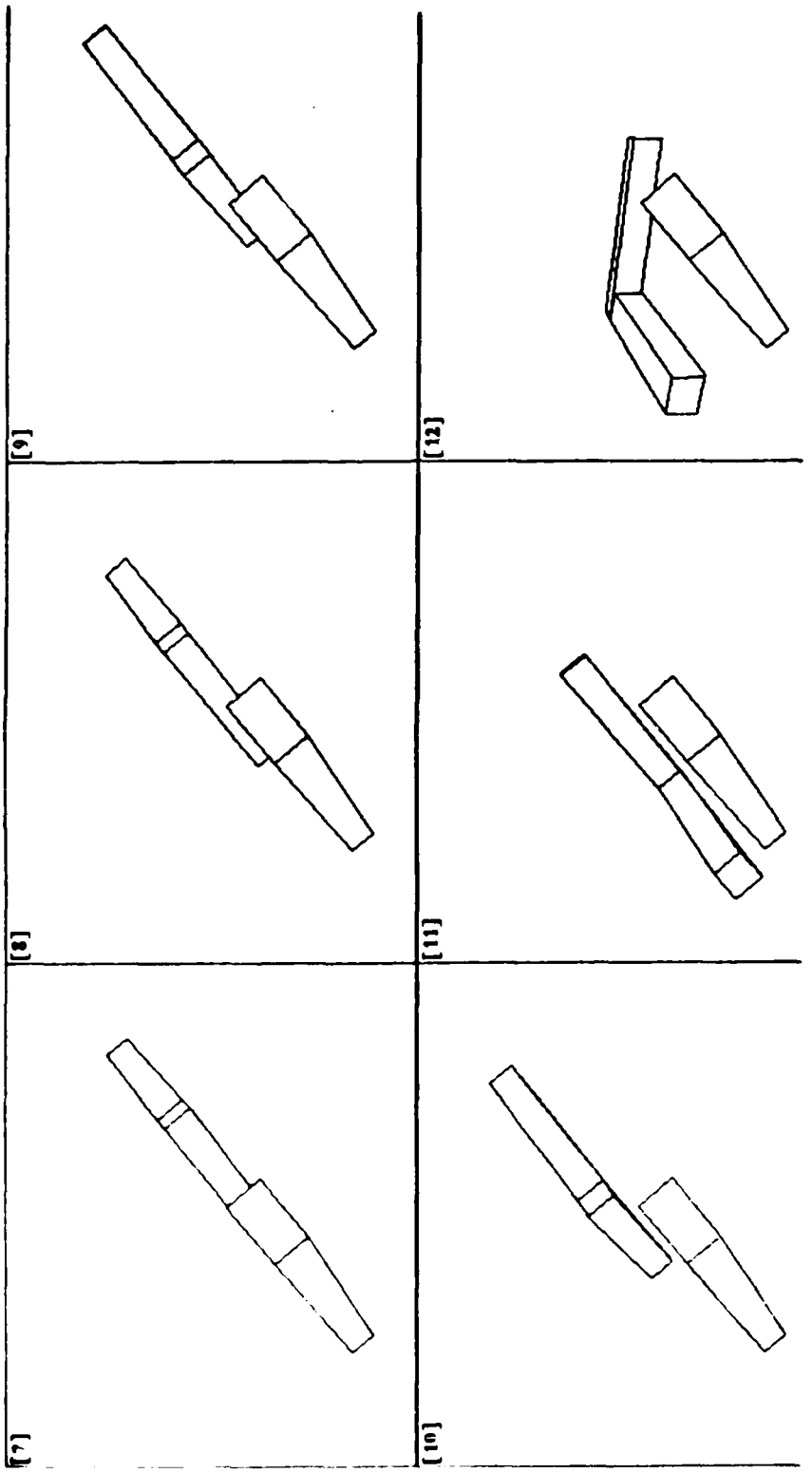


Figure 1.26 Puzzle Problem, frames (7-12), view 2

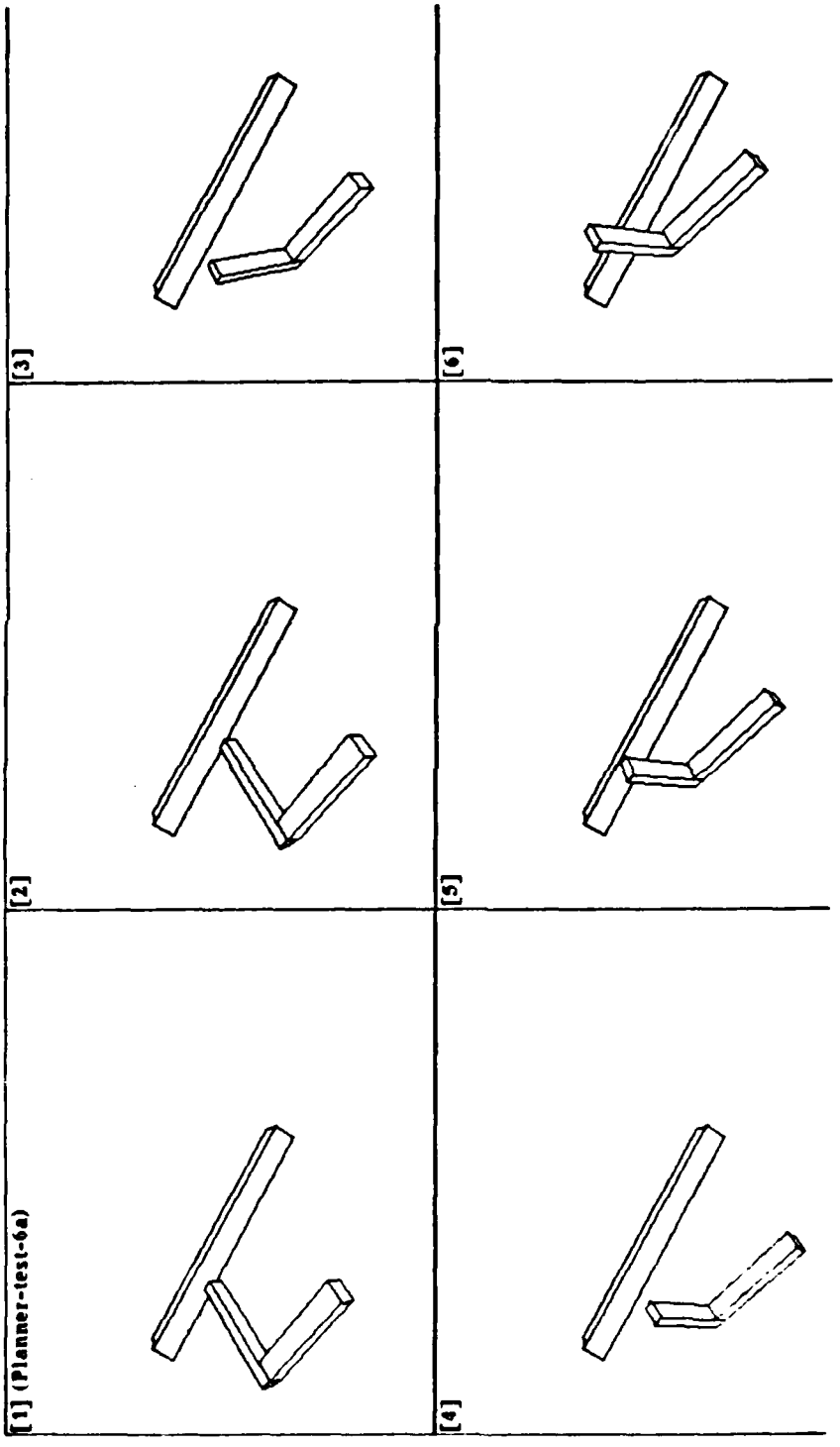


Figure I.27. Puzzle-Problem, frames (1-6), view 3

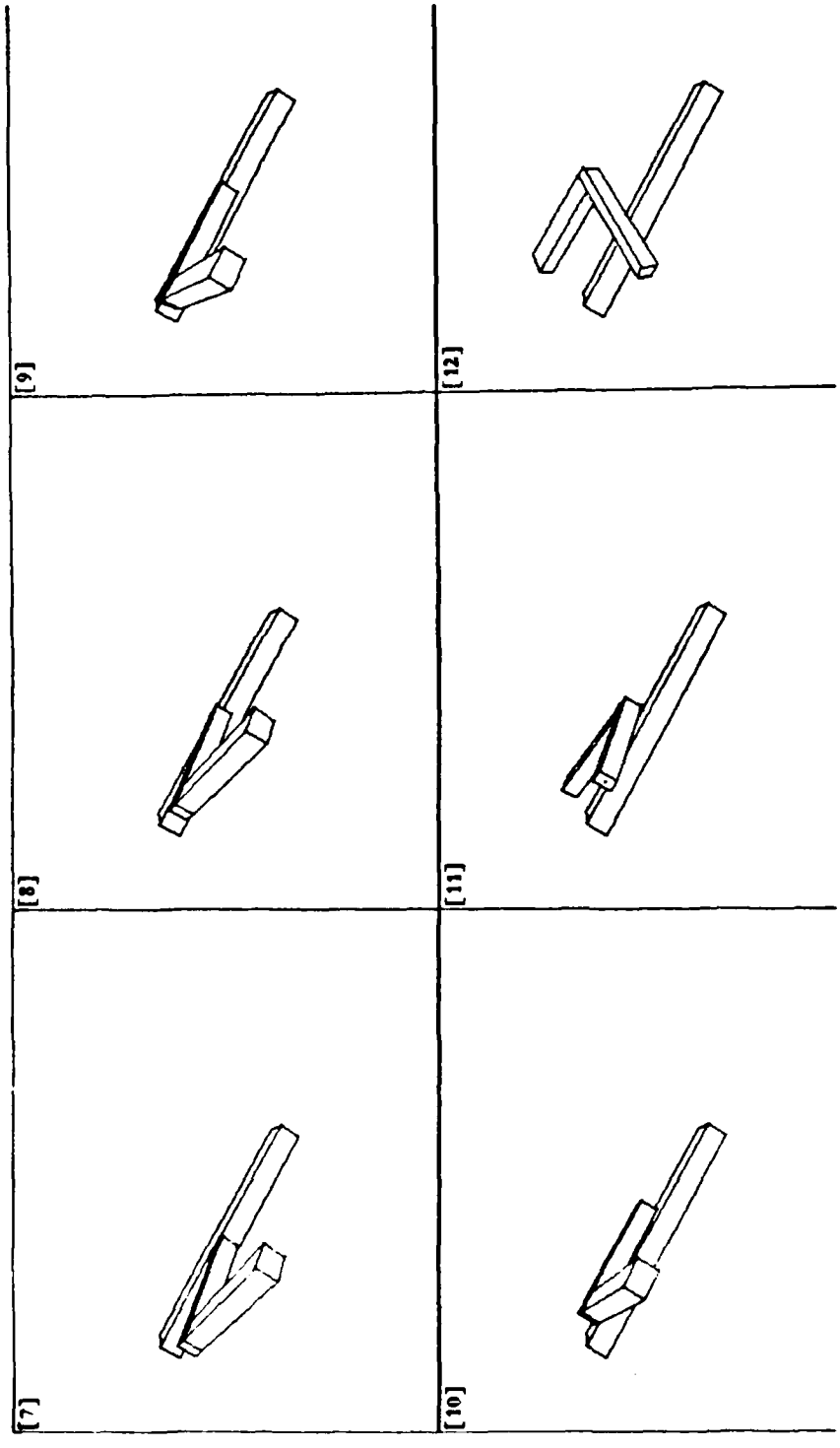


Figure 1.28. Puzzle-Problem, frames (7-12), view 3

we wish to find a path for a point in $\Re^3$, avoiding collisions with polyhedral obstacles, where each polyhedron is modeled as the intersection of a finite number of half-spaces of $\Re^3$. One solution might be to move until the point comes in contact with a polyhedron, and then to move around the obstacle by traversing the edge-graph on its boundary. (Refer to figure 1.39, ignoring the caption for now). Each arc in the edge-graph is the intersection of two surfaces bounding half-spaces. Even if the polyhedra are allowed to overlap, the technique will still work since their intersections have the same structure. Naturally, we will also need some technique for jumping from one obstacle to another.

To summarize, we can find a collision-free path for a point amidst obstacle polyhedra in $\Re^3$ through the *closure* of three operators:

The Point Navigation Operators

Operator (i) slides along the 1-dimensional edges, which lie in the intersections of the obstacle planes;
Operator (ii) slides along the two dimensional obstacle planes, which contain faces of the obstacle polyhedra;
Operator (iii) jumps from one 3-dimensional obstacle to another.

We now review the configuration space transformation. Using this transformation in its simplest form, the find-path problem in three dimensions without rotations is reduced to the point navigation problem amidst polyhedral obstacles.

1.3. Configuration Space

The *configuration* of a moving object is a vector of parameters representing its combined translation and orientation, relative to a specified coordinate system. For the classical Movers' problem in the plane, a typical configuration

$$(x, y, \theta)$$

represents a displacement (translation) of (x, y), and a rotation by θ. (For example, imagine a polygon displaced by (x, y), and rotated by θ about one of its vertices). For the six degree of freedom classical Movers' problem, a typical configuration

$$X = (x, y, z, \mathcal{R}(\Theta))$$

represents a displacement (translation) of (x, y, z), and a three dimensional rotation $\mathcal{R}(\Theta)$. The three dimensional rotation group is a three parameter family; typical representations of rotations include Euler angles, (Symon (1971)), spherical angles, and quaternions (Hamilton (1969)). For example, if the Euler angles $\Theta = (\psi, \theta, \phi)$ are employed, then they determine a 3 by 3 rotation matrix which functions as $\mathcal{R}(\Theta)$ in the rotation group. It is convenient to identify the rotation operator with its parameterization, that is, to express X as

$$X = (x, y, z, \psi, \theta, \phi).$$

Using configuration space, reasoning about the motion of a complicated three-dimensional body amongst obstacles may be transformed into reasoning about a point in a six dimensional configuration space. The transformation described by Lozano-Pérez (1983) entails "shrinking" the moving object to a point, and correspondingly "growing" the obstacles. In principle, the point may then be navigated around the grown obstacles by means of the point navigation operators (above).

In this thesis, the point navigation operators will be generalized to the six-dimensional configuration space of the classical Movers' problem.

In order to present our algorithm for planning in *C-Space*, it is necessary to review the basics. We present an introduction to representations in configuration space at two levels: first, we present an intuitive discussion. Next, we present a more detailed, slightly more mathematically-oriented exposition. For the sake of readability, there is some redundancy in the sections. Those who are encountering configuration space for the first time may wish to postpone reading the latter section for now.

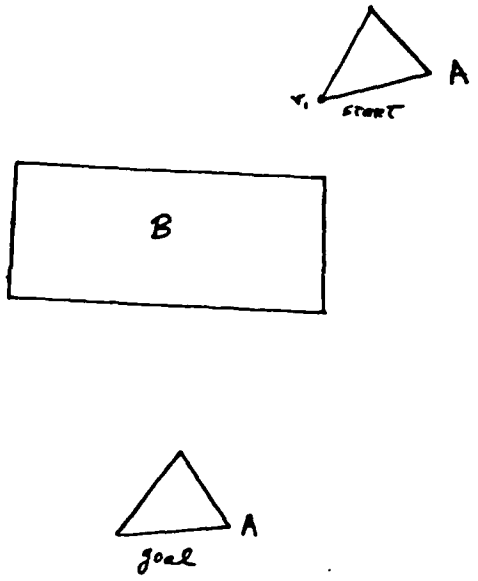


Figure 1.29. These figures show an obstacle polygon B and a moving polygon A. A must be moved around B to the goal configuration.

1.3.1. Representations in Configuration Space: An Intuitive Treatment

Figure 1.29 is an example of the classical Movers' problem in two dimensions, without rotations. A is a moving object which must be moved from the *start* configuration to the *goal* configuration, around an obstacle polygon B. The start and goal configurations may be expressed as two dimensional vectors of the form (x, y) which represent the displacement of a vertex v_1 on A from a fixed coordinate frame. The displacement is a rigid translation of the polygon A. The *C-Space* of this Movers' problem is the space of two dimensional translations, which is the same as the Cartesian plane. Lozano-Pérez (1983) demonstrated a transformation which shrinks A to the vertex v_1, while inversely growing B. The grown obstacle for B is a *C-Space* obstacle called $CO(B)$, and is shown in figure 1.31. (We will discuss the details of this transformation later). The problem of moving the polygon A from the start to the goal is transformed into the problem of navigating the point

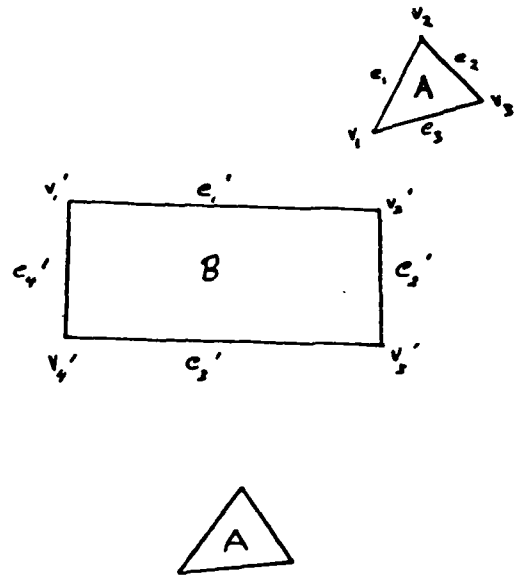


Figure 1.30. The edges and vertices of A and B.

v_1 around the *C-Space* obstacle shown in figure 1.31.

Both A and B are convex; non-convex objects are represented by overlapping unions of convex polyhedra. The *C-Space* obstacle $CO(B)$ is constructed by considering all feasible interactions of the edges and vertices of A and B. Each such interaction generates a *constraint* which is manifest as an edge of $CO(B)$. We say that an interaction between a vertex of A and an edge of B, or between an edge of A and a vertex of B, is feasible if there is some pure translation which can bring the vertex and edge in contact without causing A and B to overlap. For example, the set of all possible interactions of A and B is the union of the two cartesian products

$$\{e_1, e_2, e_3\} \times \{v_1', v_2', v_3', v_4'\}$$

and

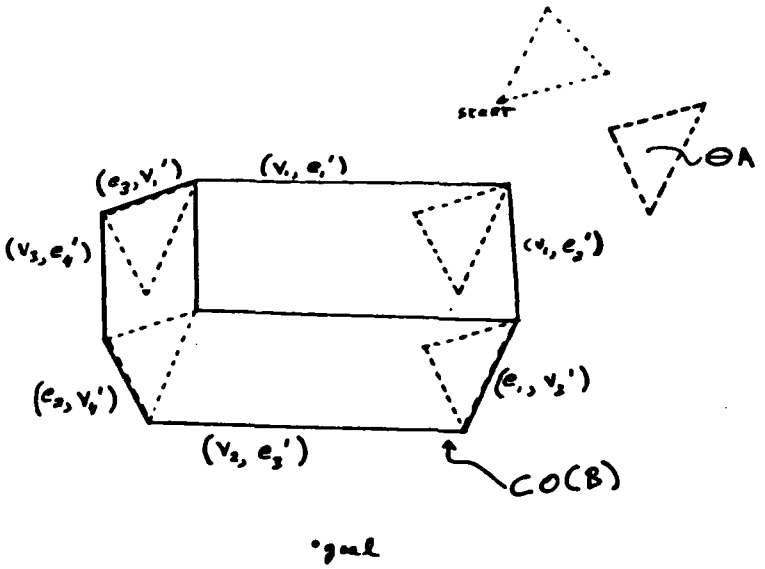


Figure 1.31. $CO(B)$, the grown obstacle for B in *C-Space*. The vertex v_1 must be navigated around $CO(B)$.

$$\{v_1, v_2, v_3\} \times \{e_1', e_2', e_3', e_4'\}.$$

However, at the depicted orientation of A, only these interactions are feasible:

$$\{(v_1, e_1'), (v_1, e_2'), (e_1, v_3'), (v_2, e_3'), (e_2, v_4'), (v_3, e_4'), (e_3, v_1')\}.$$

It is easy to visualize the translation that will bring any of these pairs into contact. Furthermore, note that (for example), no translation can bring v_3 in contact with e_2'.

Now imagine that A is allowed to rotate about v_1. At different orientations, there will be different sets of feasible contacts. We say that the constraints associated with feasible contacts are *applicable* constraints. It should be clear that at any given

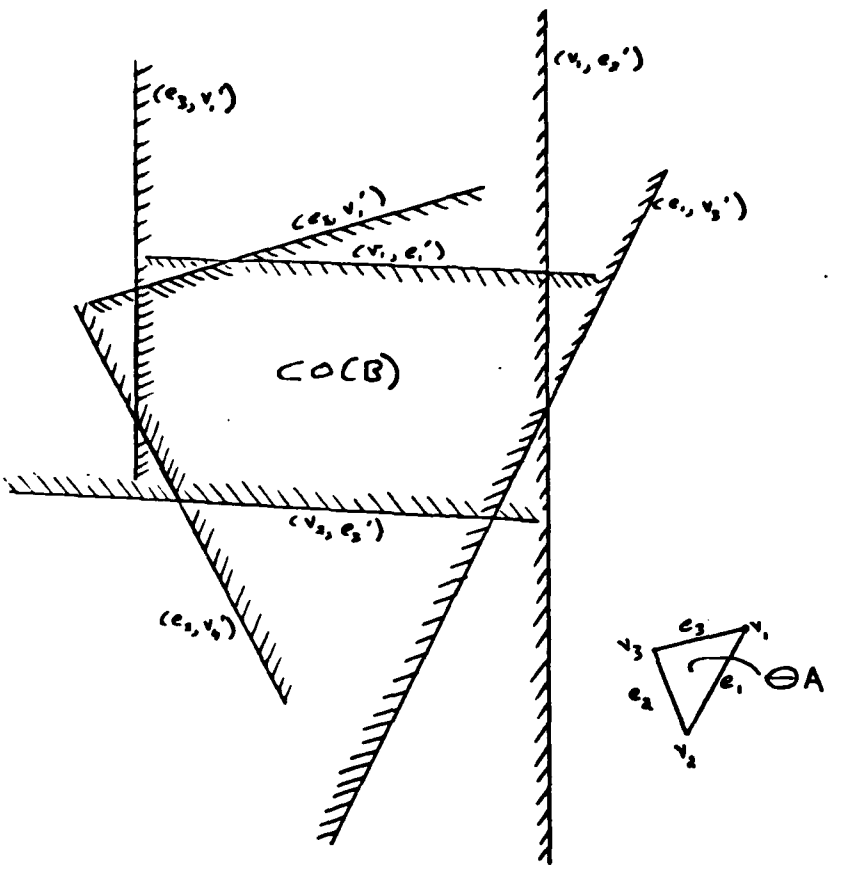


Figure 1.32. $CO(B)$ can be represented as the intersection of 7 half-spaces, whose boundaries contain edges bounding $CO(B)$.

orientation, only certain constraints will be applicable. The orientations for which a given constraint is applicable form its *applicability region*. The applicability regions for each constraint in this problem are angular sectors of the form $[\theta_l \leq \theta \leq \theta_u]$. This simply means that there exists a range of angles in which a particular contact is feasible. This range of angles may be geometrically visualized as a sector of a circle.

When A is allowed to rotate, the geometry of $CO(B)$ changes as θ varies, and as the set of applicable constraints changes. As the edges and vertices of A rotate about v_1, the constraints they generate sweep out ruled surfaces which bound $CO(B)$ in a three dimensional *C-Space*. (The *C-Space* is three dimensional, since A now has three degrees of freedom: x, y, and θ). At any fixed orientation θ_0, an x-y slice of $CO(B)$ is a polygon, called $slice(CO(B), \theta_0)$. Figure 1.31 shows such a slice at the depicted orientation of A. With each edge of $slice(CO(B), 0)$ there is an

associated half-plane containing $slice(CO(B), \theta)$, whose boundary contains the edge (see figure 1.32). The intersection of these half-planes is exactly $slice(CO(B), \theta)$. As θ changes, different half-planes are used to construct $slice(CO(B), \theta)$. By (1) deriving the line equation of the boundary of these half-planes in terms of the orientation θ, and (2) by determining the applicability region for each half-plane constraint, we can characterize slices of the *C-Space* obstacle $slice(CO(B), \theta)$ as θ changes. Thus we can characterize the three dimensional *C-Space* obstacle $CO(B)$. This representation may be used to develop planning algorithms for the Movers' problem with two translational and one rotational degrees of freedom (see Brooks and Lozano-Peréz (1983)).

In this thesis we develop such a representation for the six degree of freedom Movers' problem. There are several problems which must be solved. Because the structure of the rotation group is more complicated in three dimensions, the applicability regions for constraints in a six degree of freedom *C-Space* are geometrically much more complicated. While in two dimensions the applicability regions may be visualized as sectors on a circle, in three dimensions they are complicated three dimensional manifolds on the projective 3-sphere. It is important to characterize these regions, since they specify where a constraint is applicable. We will discuss some of the other problems presently.

Generalizing the Point Navigation Operators Requires Solving Representational Questions

In order to generalize the point navigation operators to the *C-Space* of the classical Movers' problem, we must be able to characterize the surfaces of *C-Space* obstacles, and the intersections of these surfaces. The first two operators, then, must slide along the C-surfaces and their intersections. In the next section, we discuss some of the representational issues involved in developing such operators. For example, when rotations are allowed, the C-surfaces are curved. In the six-dimensional space of the classical Movers' problem, each C-surface is a five-dimensional submanifold of *C-Space*, and the intersection of two such surfaces is a four-dimensional manifold. Thus it is possible to slide along such an intersection with four degrees of freedom.

1.3.2. Representations in Configuration Space: A More Formal Treatment

In this section, we present a somewhat more abstract formulation of representational issues in *C-Space*. Some readers may wish to postpone reading this section until later. We will proceed as follows: first, we will outline an important representational question which must be solved in this thesis. Next, we discuss how to represent volumes (such as *C-Space* obstacles) in *C-Space*. In the course of this discussion, several terms will be defined in context by means of intuitive descriptions. At the end of this section, under the heading *Working Definitions*, we will summarize and formalize the definitions to the extent that will be required in chapters 1 and 2.

The Domain Question

Until now, geometric planning problems with more than three degrees of freedom have proved resistant to solution.[1] In this thesis we provide such an algorithm for find-path with six degrees of freedom (the classical Movers' problem). The resistance of these problems has largely been due to unresolved mathematical issues and questions relating to the structure of configuration space and to the nature of *C-Space* constraints, (although for fine-motion and planning with uncertainty there are of course additional issues).

One fundamental theoretical problem for high-dimensional configuration spaces may be stated as follows: in a configuration space C with rotations, each *C-Space* obstacle may be represented by the intersection of a finite number of half-spaces. Each half-space, in turn, is defined by a real-valued *C-function* on *C-Space*. For example, the half-space might be the set of configurations where the C-function is negative. However, each C-function is a *partial* function on C, whose domain is a complicated region in *C-Space*. This greatly complicates the representation for *C-Space* obstacles and C-surfaces (see figure 1.36). Moreover, until now the domains of the C-functions were unknown for all but the one-dimensional rotation group. One of our first tasks will be to derive the domains of all C-functions for the classical Movers' problem with six degrees of freedom.

[1] However, previous work has provided an existence proof of a polynomial time algorithm for certain spatial planning problems. In addition, there are approximate algorithms for some of these problems. See our review of previous work.

There are several related problems, for which we also present solutions. This allows us to construct a complete geometric representation for the configuration space of the classical Movers' problem with six degrees of freedom. This representation impacts all the geometric planning problems we have discussed, and extends naturally to Cartesian Manipulators.

Representing Volumes in Configuration Space

The dimensionality of configuration space is the number of degrees of freedom in the parameter space, i.e., the number of degrees of freedom available to the moving object(s). Thus the classical Mover's problem in the plane has two translational and one rotational degrees of freedom, while in three dimensions it has three translational and three rotational degrees of freedom. The configuration spaces for these problems are three and six dimensional manifolds, respectively. As the number of degrees of freedom increases, a geometric planning problem becomes harder. There are several reasons for this. First of all, when rotations are allowed, configuration space ceases to be Euclidean, and the C-surfaces become curved. Furthermore, the non-commutativity and multiple-connectedness of the three-dimensional rotation group are classical difficult issues in mathematics. In addition, it can be shown that the computational complexity of spatial planning grows exponentially with the dimensionality of the *C-Space*.

A fundamental issue for geometric planning algorithms is: how should *C-Space* obstacles and surfaces be represented and computed?

A volume in a configuration space C may be represented by the intersection of a finite number of half-spaces (see figure 1.33). Each half-space may be defined via some smooth, real-valued function function on C,

$$f_i : C \rightarrow \Re.$$

For example, (fig. 1.34) suppose $f_i(x, y) = ax + by + c$, for some constants a, b, and c. The kernel of f is the line where $f(x, y) = 0$. The halfspace h_i^- is the portion of the plane where $f(x, y)$ is negative. C-functions such as f_i arise in the two dimensional Movers' problem, at a fixed orientation.

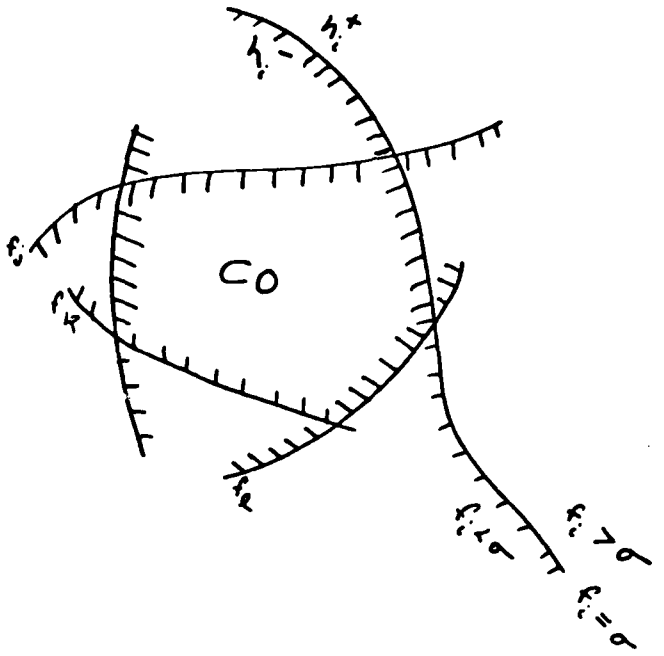


Figure 1.33. The region CO is the intersection of the half-spaces h_i^-, h_j^-, h_k^-, and h_l^-.

In general, the (closed) half-space h_i^- is the set of all points in C where f_i is negative-valued or zero:

$$h_i^- = \{v \in C \mid f_i(v) \leq 0\}.$$

The common intersection of a number of such half-spaces can yield a volume in C. Lozano-Pérez (1983) showed how *C-Space* obstacles can be modeled in this manner, and gave the form of the functions f_i. Note that each C-surface lies within the kernel of some constraint f_i. Fine-motion strategies and algorithms for planning with uncertainty need to compute the normals and tangent spaces to these C-surfaces. The normal can usually be derived from the gradient ∇f_i (this requires placing an appropriate Riemannian metric on the tangent space). When a real-valued function f_i on configuration space is used to describe constraints in

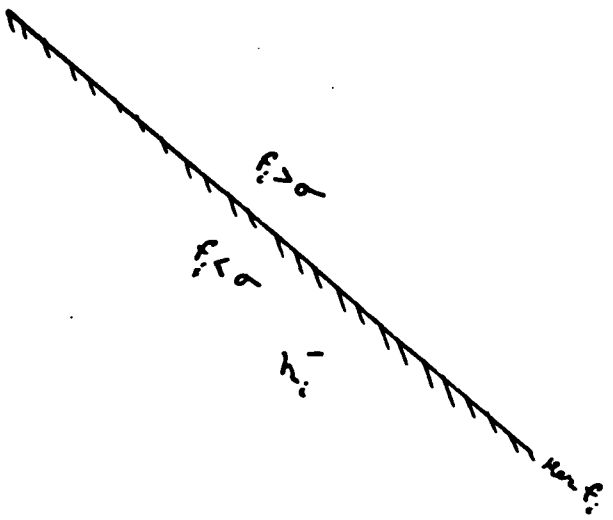


Figure 1.34. Example of a C-function f_i in the plane.

that *C-Space*, (i.e., *C-Space* obstacles), we call it a *C-function*. The form and interpretation of C-functions are presented later.

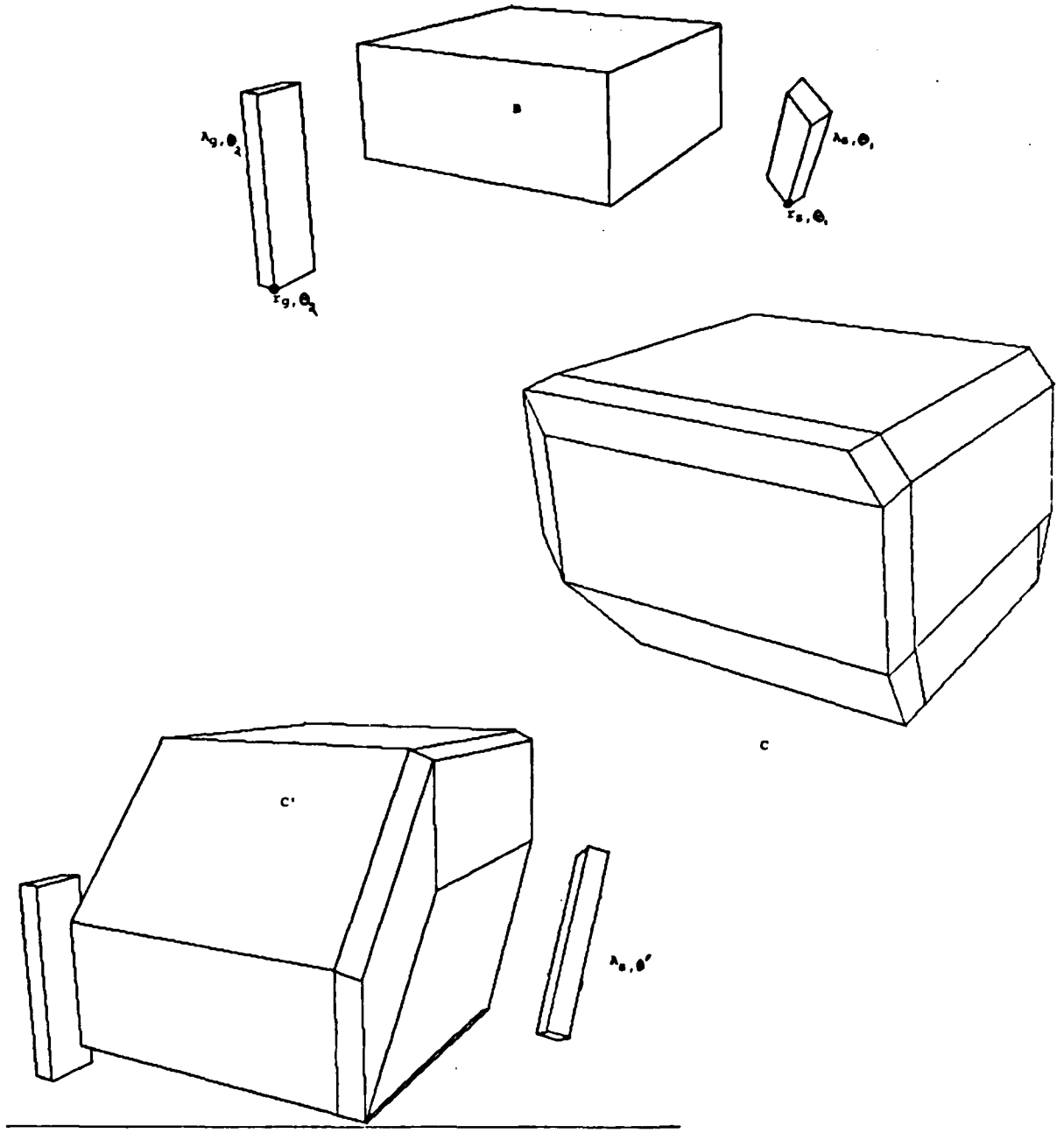


Figure 1.35. Illustration of the classical Mover's problem in three dimensions. B is an obstacle, and A is an object which must be moved around B. A_{s,Θ_1} shows A in the start configuration, and A_{g,Θ_2} shows A in the desired goal configuration. C is the polyhedron which is the *C-Space* obstacle from B for A at orientation Θ_1. At orientation Θ', the *C-Space* obstacle from B is a different polyhedron, which we show as C'.

Example: (See figure 1.35). Consider the Movers' problem for a three dimensional polyhedron A which can translate but not rotate amidst polyhedral obstacles. The configuration space for A is a three-dimensional vector space of translations, which it is convenient to identify with $\Re^3$. Each constraint f_i will be linear on $\Re^3$, and the kernel of f_i is a plane. Each such plane bounds a *C-Space* obstacle (such as C or C' in figure 1.35). The *C-Space* obstacles are possibly overlapping polyhedra in $\Re^3$. The find-path problem in the transformed space is that of navigating a point past the union of these *C-Space* polyhedra (see figure 1.39).

In this formulation there is a fundamental underlying assumption: f_i is a *total function* on $\mathcal{C}$, that is, the domain of f_i is all of $\mathcal{C}$. In the example this is not a problem, since the domain of each linear C-function is the entire space. A function whose domain is a subset of $\mathcal{C}$ is called a *partial function* on $\mathcal{C}$. When rotations are allowed, C-functions become partial functions.

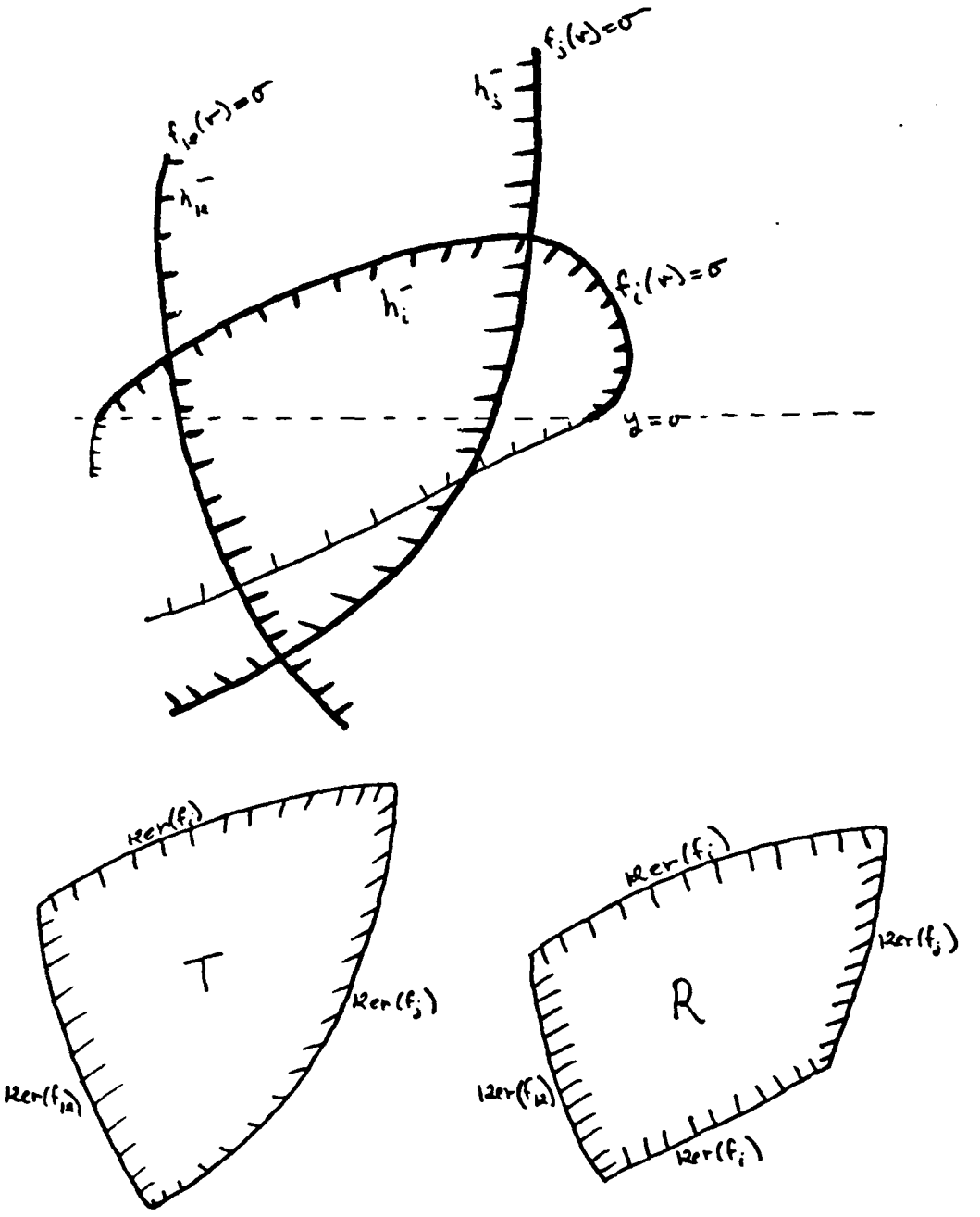


Figure 1.36. The functions f_i, f_k, and f_k are used to describe the half-spaces h_i^-, h_j^-, and h_k^-. If all functions are *total* functions on the plane, then the intersection of the corresponding half-spaces will be the the *rectangloid* region R. However, suppose that f_i is a *partial function*, whose domain is restricted to the half-space where y is positive. We say that f_i is *not applicable* below the line $y = 0$. Furthermore, we assume that points outside the domain of f_i are within h_i^-. In this case, the intersection $h_i \bigcap h_j^- \bigcap h_k^-$ is triangloid region T.

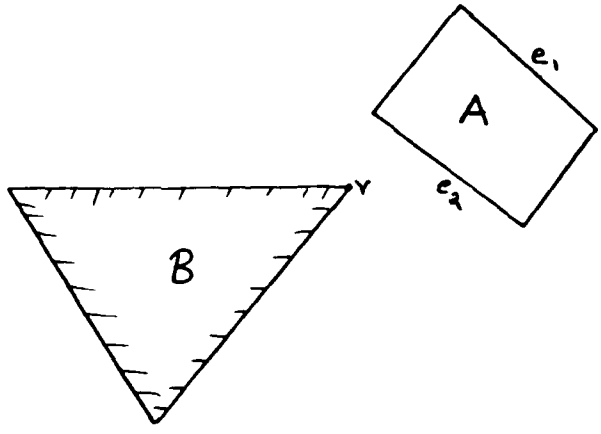


Figure 1.37. The two-dimensional classical Movers' problem: An obstacle polygon B and a moving polygon A. A is shown at a particular orientation, θ_1.

Why do *C-Space* constraints become partial functions when rotations are introduced? Consider the classical find-path problem in two dimensions, for a moving polygon A which can translate and rotate in the plane (see figure 1.37). A configuration of A may be represented by three parameters, (x, y, θ). The surfaces of the *C-Space* obstacle for B arise from each of the feasible contacts (or *interactions*) between the edges and vertices of A and B. Thus the constraint functions (which we have been calling f_i) are defined by considering pairwise interactions of edges and vertices of A with vertices and edges of B. Every such pair such as e_a and v_b will generate a smooth, real-valued C-function f_{e_a,v_b} on configuration space.[2] Each constraint is designed such that their conjunction enforces a disjointness criterion for A and B. However, not all interactions are possible at any given orientation. For example, at the depicted orientation θ_1 of A, edge e_2 can interact with vertex v, but edge e_1 cannot: at orientation θ_1, no translation can bring e_1 in contact

[2]For the form of the C-functions, see chapters 3 and 4.

with v while maintaining the disjointness of the interiors of A and B. We say the associated C-function $f_{c_1,v}$ is *not applicable* at orientation θ_1. In other words, no configuration

$$(x, y, \theta_1) \in \Re^2 \times \{\theta_1\}$$

is in the domain of $f_{c_1,v}$. Each constraint f_{c_a,v_b} is applicable only at certain orientations, and hence each can be considered a partial function on the *C-Space*.[3]

In three dimensions (see figures 1.35 and 1.39), the surfaces of the *C-Space* obstacle for B arise from each of the feasible contacts between the vertices, edges, and faces of A and B. By analogy with figure 1.37, it is clear that not all of these interactions are feasible at any given orientation. Thus the C-functions describing C-surfaces for spatial planning with six degrees of freedom must also be partial functions.

Working Definitions: Review and Summary

We now summarize and formalize the key definitions and concepts required in this chapter and the next:

Configuration Space: (Formal definition) Configuration space is the product space of the space of translations and the space of rotations for an object. In three dimensions, the space of translations is Euclidean 3-space $\Re^3$ and the space of rotations is the 3-dimensional rotation group or *Special Orthogonal Group*, $SO(3)$. $SO(3)$ is isomorphic to the intersection of the Special Linear Group (the set of all real 3×3 matrices with determinant 1) and the Orthogonal Group (which may be thought of as the set of matrices with orthonormal rows and columns). The orthonormality of rows and of columns are equivalent conditions. $SO(3)$ is isomorphic to P^3, the 3-sphere S^3 with opposite points identified. (P^3 is also known as the *projective 3-sphere*). S^3 is isomorphic the group of unit quaternions. For the classical Movers' problem we will employ configuration space, $\Re^3 \times SO(3)$. We will denote the classical Movers' Problem with three translational and three rotational

[3]See Brooks and Lozano Peréz (1983) for a discussion of the domains of C-functions for the two dimensional find-path problem with rotations.

degrees of freedom as $6DOF$. In practice, we will represent rotations as members of a three-parameter family (for example, Euler Angles), but we must keep in mind that they parameterize an isometry and that $\Re^3 \times SO(3)$ is not a vector space. If the Euler angles (ψ, θ, ϕ) are employed to represent the orientation of a rigid polyhedral body, a typical configuration X in $\Re^3 \times SO(3)$ has the form

$$X = (x, y, z, \psi, \theta, \phi).$$

We will sometimes adopt the notation

$$X = (r, \Theta),$$

where r denotes a three-dimensional translation vector, and Θ some three dimensional rotation. This second notation is independent of the particular representation chosen for rotations; the first isn't. If Euler angles are employed, we may think of Θ as the "vector" of Euler angles, (ψ, θ, ϕ).

C-Space Obstacle: (Informal definition) Configuration space obstacles are (possibly overlapping) six dimensional manifolds (with boundary) which correspond to sets of configurations that would cause collisions of the moving object with real space obstacles.

Free space: The free space is that subset of *C-Space* which lies within no *C-Space* obstacle. The free space will be denoted F.

Applicability Set: (Informal definition) Refer to figures 1.29-32 and 1.33-38, and recall that *C-Space* obstacles are represented by the intersection of a finite number of half-spaces. (To be formal we should call them half *hyperspaces*). The boundary of each half-space is a C-surface, and contains a boundary patch of the *C-Space* obstacle. Each C-surface S may be expressed as the kernel of a real-valued function f on *C-Space*. The *C-function* f is negative on the obstacle side of the half-space *C-Space* obstacle, and positive on the other half. In the literature C-functions have been called *constraints*, since they express constraints on the possible motions for

an object. A surface parallel to S is called a level C-surface, and represents the set of configurations where f has a certain fixed value. This value is termed the level of the level C-surface. The boundary of the *C-Space* obstacle is a special case of level C-surface, where the level is zero. We have seen that at any given orientation, only certain C-functions (and their associated C-surfaces) are applicable. This is because only certain contacts are feasible between the faces, edges, and vertices which generate the C-functions. We call this set of C-functions the *applicability set.* For example, in 1.29-32, at the depicted orientation the applicability set is

$$\{(v_1, e_1'), (v_1, e_2'), (e_1, v_3'), (v_2, e_3'), (e_2, v_4'), (v_3, e_4'), (e_3, v_1')\}.$$

(Actually, the applicability set is the set of C-functions generated by these vertex/edge and edge/vertex pairs, but since there is a one-to-one correspondence between the generator pairs and the C-functions, we can write it this way). In later chapters, we will demonstrate algorithms for computing the applicability set, and for decomposing rotation space into regions where the applicability set is invariant.

Redundant and Non-redundant Constraints: (Informal definition). If a configuration X is in free space, the set of constraints which is (locally) relevant to motion planning from X is a subset of the applicable, positive-valued C-functions at X. However, the value of a C-function does more than merely indicate which side of a C-surface X is on. A C-function's value represents the translational distance to that surface. Thus, C-functions provide a collection of pseudo-metrics on *C-Space*. Using these metrics, it is possible to order C-surfaces by their closeness to a configuration X (simply sort the C-functions on their value at X). We say that a C-surface is *redundant* if it is subsumed by a nearer, intervening constraint. In figure 1.38, for example, f and g are non-redundant constraints at X, but h is redundant since it is subsumed by f. It is useful to determine the set of non-redundant constraints at X since this is the smallest set of constraints that are locally relevant to motion planning. We provide a formal definition of redundancy in chapter 3.

Robot, Moving Object, and Piano: All of these terms have been employed in the literature to refer to the moving object for which a collision-free path must

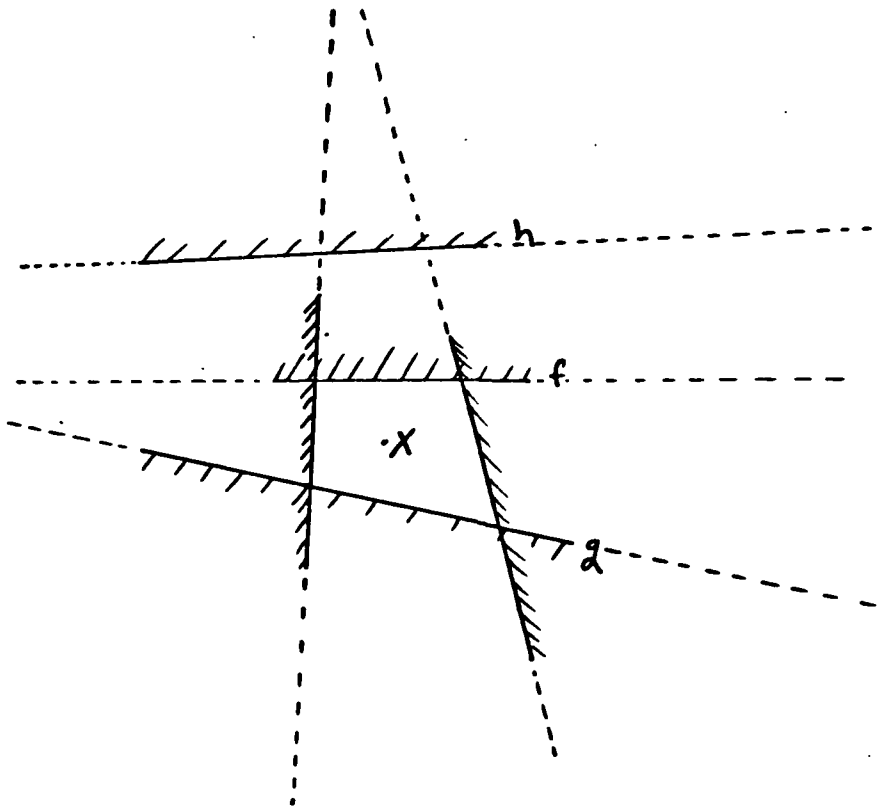


Figure 1.38. h is a redundant constraint.

be found. Our moving object is modeled as the possibly overlapping union of a finite set of convex polyhedra. The union is rigid, but not necessarily convex or connected. The moving object has three translational and three rotation degrees of freedom. To avoid monotony, we may employ the term *robot* to refer to the moving object. The terminology is justified in part by the fact that our algorithm extends straightforwardly to Cartesian manipulators.

The Research Agenda of this Thesis

A Brief Outline

I. Computational Theory

Paths can be found in *C-Space* by the closure of three operators:
(i) slides along 1- to 4-dimensional intersections of level C-surfaces;
(ii) slides along 5-dimensional level C-surfaces;
(iii) jumps between 6-dimensional obstacles.

II. Representation and Algorithm

Search Algorithm Employing the Three Operators

⇑

Solve the Intersection Problems.
Develop a represention for the intersection manifolds.

⇑

Solve open questions about the structure of 6DOF constraints. Derive and represent structural properties of the constraints, for example, the domains of defining partial functions. Develop decomposition algorithms.

III. Implementation

Implement the 6DOF planner.

IV. New Theoretical Results

The structure of 6DOF constraints: Theorems on the domains and domain topology of the defining partial functions.
Theorems on the applicability decomposition.
The C-Voronoi Diagram (CVD).
The Equivalence Theorem for intersection manifolds and the CVD.
Criteria for designing/integrating local and global planning algorithms.

1.3.3. Generalizing the Point Navigation Operators

Consider a three-dimensional configuration space containing smooth, curved *C-Space* obstacles. Observe that the point navigation operators will work even if the surfaces are curved and complicated, as long as we can find their intersections. For the two-dimensional Mover's problem (for a polygon allowed to rotate and translate in the plane), we employ a configuration space $\Re^2 \times S^1$. $\Re^2$ is the space of two dimensional translations, and S^1 is the unit circle, on which one-dimensional rotations may be represented. C-functions are of the form $f_i : \Re^2 \times S^1 \rightarrow \Re$ and are valid within some sector A_i of S^1. A C-surface is the set of configurations where f_i is zero. Although these C-functions are complicated expressions containing trigonometric terms of the form $x\cos\theta$ and $y\sin\theta$, it is possible to solve two such C-surfaces simultaneously to obtain an intersection curve in $\Re^2 \times S^1$ which is parametric in θ (these intersection manifolds are derived in chapter 4). The analogy to navigating a point through a polyhedral environment should now become clear: the faces of the polyhedra correspond to C-surfaces in $\Re^2 \times S^1$ and the edge-graphs to the graph of C-surface intersections. By searching the graph of C-surface intersections we can find a path in configuration space, *if one exists.*

Planning in a Six Dimensional *C-Space*

Our planner for a six dimensional *C-Space* is based on the idea of moving along the intersections of level C-surfaces in free space, parallel to the boundaries of *C-Space* obstacles. In the example above, the coincidence between the dimensionality of configuration space and Euclidean space was serendipitous: edges on polyhedra corresponded to curves in $\Re^2 \times S^1$, and faces to 2-dimensional surfaces. However, in a six dimensional *C-Space*, the C-surfaces are 5-dimensional and their intersections are 4-dimensional sub-manifolds. Intuitively this means that the set of possible motions while complying with two constraints is a four-parameter family.

Our idea is as follows: Suppose we could slide along C-surfaces (see figure 1.39). In addition, suppose we could intersect C-surfaces to construct a lower dimensional manifold in *C-Space* which contained paths along (or around) the boundary of *C-Space* obstacles. By sliding along C-surfaces, and by sliding along the intersection of C-surfaces, we should be able to devise an algorithm which can circumnavigate

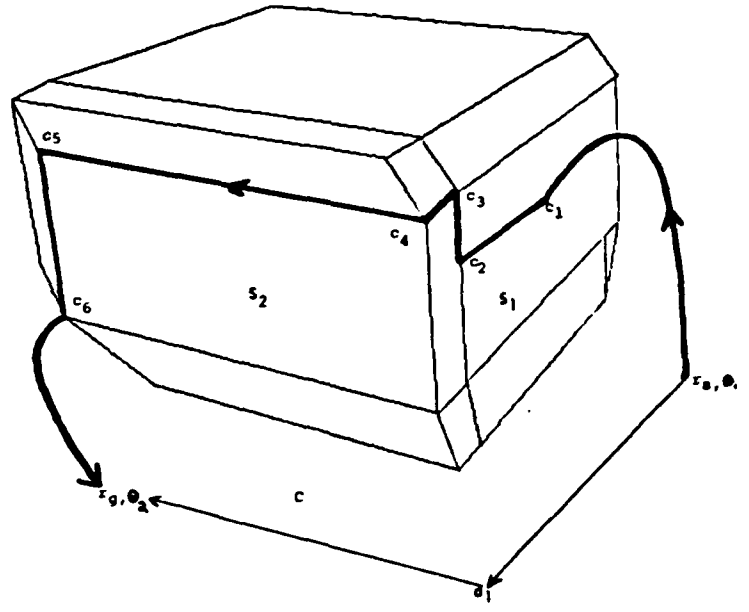


Figure 1.39. We can represent the configuration of a polyhedron A by a pair, (T, Θ), where T is a translation of A and Θ is a rotation of A. The problem of moving A from configuration (r_s, Θ_1) to (r_g, Θ_2) is transformed to the problem of navigating a *configuration point*, r, past C, which is the *C-Space* obstacle due to B. S_1 and S_2 are C-surfaces bounding C. The configurations c_i lie on the boundary of C, while d_1 is in free-space. Two trajectories around B are shown. Note that the path segments $(c_6, (r_g, \Theta_2))$ and $(d_1, (r_g, \Theta_2))$ must also include a rotation. (The actual reference point is at the centroid of A, but for the purposes of exposition, we have placed it at a vertex as shown).

C-Space obstacles. (Of course, we also need a way to plan motions which "jump" from one obstacle to another).

Example: Figure 1.39 shows how such a planning algorithm might work. The planner moves through free-space from (r_s, Θ_1) until it strikes a C-surface S_1 at c_1. From c_1 a path is found towards c_2 sliding along the C-surface S_1. We say the planner slides along S_1 to c_2. Configuration c_2 lies on an intersection manifold of the C-surfaces S_1 and S_2. The path segment (c_2, c_3) slides along this intersection manifold, which lies on the boundary of C. A path $(c_2, c_3, c_4, c_5, c_6)$ is planned along the graph of intersection manifolds on the boundary of C. From c_6 we leave the boundary of C, and translate and rotate through free-space to (r_g, Θ_2). This path

is an idealized example of planning along C-surfaces and intersection manifolds of C-surfaces. The implemented planner finds a path similar to $((r_s, \Theta_1), d_1, (r_g, \Theta_2))$ (see chapter 2). The path segment $((r_s, \Theta), d_1)$ is parallel to the C-surface S_1, and we say that it slides along a *level C-surface* for S_1. The path segment $(d_1, (r_g, \Theta))$ is along a level C-surface for S_2. These level C-surfaces intersect along a manifold in free-space containing d_1 (imagine extending the faces S_1 and S_2 beyond the boundary of C until they intersect).

We will derive the necessary mathematical theory and tools relating to C-surfaces and their intersection manifolds, and present algorithms for moving and planning paths in *C-Space*. Some of the issues we will address include:

(i) What is an appropriate representation for constraints in a six dimensional *C-Space*?

(ii) How do we plan motions using constraints whose domains change with the motions?

(iii) How can trajectories in *C-Space* be intersected with C-surfaces whose domains change along the trajectory?

(iv) How can intersection manifolds be constructed in *C-Space*?

(v) How are motions planned which slide along C-surfaces and intersection manifolds?

It is useful to develop a terminology for evaluating algorithms and representations for geometric planning problems. An algorithm employing an *approximate representation* does not characterize the constraints exactly. A *complete* algorithm (for a given resolution) is guaranteed to find a solution if one exists (at that resolution). In general, construction of a complete algorithm mandates the employment of a complete representation. A *brute-force* algorithm tries to find a solution through exhaustive search. Heuristic algorithms fall into two (overlapping) classes: heuristically complete, and heuristically fast. See also the review of previous work (below).

The implemented algorithm we present for the classical Mover's problem with six degrees of freedom employs a complete representation of the configuration space constraints, and a complete search algorithm (for a given resolution).

1.4. Local versus Global

Local algorithms for the find-path problem examine local constraints in some neighborhood of real space or in *C-Space*, and propose motions based on the geometry of the neighborhood. Typically, local algorithms are implemented as searches, and the examination of constraints near a search node leads to the selection and application of some local operator to move the robot in space. For example, in our algorithm for the six degree of freedom Movers' problem, the local constraints correspond to the geometric structure of C-surfaces in a neighborhood, and local operators consist of motions along C-surfaces.[1] In general, a local planning algorithm will be complete if (1) the closure of the local operators is complete for the arcwise-connected components of *C-Space*, and (2) each local operator attempted ensures that a collision-free path exists between configurations in the search.

However, our observation has been that in general, even complete local algorithms can get lost examining irrelevant local constraints. In particular, without adequate knowledge of the connectivity of a workspace and the classes of paths it contains, such methods may choose impossible or ill-advised candidate paths: hence they may take a long time to converge.

A *global* find-path algorithm attempts to construct a model of the connectivity of the workspace. We believe that the connectivity of configuration space can be inferred from the connectivity of real space. Good hypotheses about the *channels*, or classes of paths through free-space can serve as guidance for a more detailed method. While there exist several proposals for global approaches to the Movers' problem, in Donald (1983a) we attempt to formalize criteria for the design of such algorithms. A global planner based on these criteria was implemented, and coupled with a complete local algorithm to form an integrated planning system.

Channels are an encoding of free-space corresponding to the classes of paths within an environment. An implementation exploiting this global model of the connectivity of free-space has been able to solve two dimensional find-path problems in several minutes which formerly took many hours. The algorithm is essentially

[1] This example is illustrative and typical of the local constraints and operators. The implemented planner is more complicated, as we shall see.

a problem-solving strategy using a homeomorphic reduction of the search space. See Donald (1983a) for a description of the channel algorithm. In appendix III, we discuss the design and integration of local and global planning algorithms in $\Re^3 \times SO(3)$.

1.5. Review of Previous Work

1.5.1. Introduction

In this section we review previous work on geometric planning problems. We also give a formal characterization of completeness for the spatial planning problems. A survey of robotics issues in robot motion planning can be found in Brady, *et al.* (1980). For related work on the mover's problem, see Brooks (1983a), Lozano-Pérez (1981, 1983), Lozano-Pérez and Wesley (1979), Brooks and Lozano-Peréz (1983), Schwartz and Sharir (1982a), Reif (1979), Moravec (1979), Udupa (1977) and Hopcroft and Wilfong (1984). Wingham (1977) and Popplestone, Ambler, and Bellos (1980) consider related issues in geometric planning problems. Some issues in automated structural design are addressed in Donald (1983b). For a review of geometric modeling techniques, see Baer, Eastman, *et al.* (1979) and Requicha (1980).

1.5.2. Complexity-Theoretic Results

In seminal work on the complexity of the Movers' problem, Reif (1979) has shown that the motion planning problem for a robot with an arbitrary number of degrees of freedom in the form of arm-like linkages is *P-Space*-hard. Hopcroft, Joseph, and Whitesides (1982) have shown similar results for planar manipulators with n linkages. In general it has been found that with n degrees of freedom, the problem is *P-Space*-hard. Happily, Schwartz and Sharir (1982a) have demonstrated the existence of a polynomial-time algorithm for the Movers' problem with fixed degrees of freedom, where the size of the problem is measured in the number of obstacle faces in the environment. The algorithm of Schwartz and Sharir (1982a) for the classical Movers' problem is unfortunately of time complexity

$$O\left(n^{2d+6}\right)$$

where n is polynomially dependent on the number of faces in the environment, and d is the number of degrees of freedom. For 6 degrees of freedom this becomes $O(n^{4096})$. Hence it serves chiefly as an existence proof for a polynomial-time algorithm.

The coordinated motion problem has also been given some attention: Schwartz and Sharir (1982b) address the problem for 2 and 3 circular bodies moving amidst polygonal obstacles in the plane. The coordinated motion system has degrees of freedom equal to the sum of the degrees of freedom of the moving bodies. These results lead us to expect exponential behavior from all motion-planning algorithms as the number of degrees of freedom grows. For these reasons we will confine ourselves to the classical Movers' problem, which has 6 degrees of freedom in 3-dimensional space.

Reif (1979) also sketches a polynomial-time algorithm for the classical Movers' problem, but it appears incomplete in that it ignores constraints arising from the interactions of faces of the moving object with vertices of obstacles, and does not consider edge-edge interactions in 3 dimensions.

1.5.3. Work in Computational Geometry and Robotics

The foundations of our approach lie in Lozano-Pérez (1981, 1983), Lozano-Pérez and Wesley (1979) and Schwartz and Sharir (1982a). The problem of moving a complex polyhedral object among obstacles is transformed to the problem of finding a path for a point in a high-dimensional configuration space.

Brooks and Lozano-Peréz (1983) have implemented a general path-finding algorithm for a polygonal object in the plane with two translational and one rotational degrees of freedom. Their planner uses hierarchical subdivision of the 3-dimensional configuration space $\Re^2 \times S^1$. The subdivision algorithm has been specialized to the particular geometry of the Movers' problem in $\Re^2 \times S^1$ and while in principle it is extensible to the 6 degree of freedom problem, its space-complexity in high dimensions is likely to be unattractive. A problem with the hierarchical subdivision strategy is that it has trouble exploiting coherence in *C-Space*. Its spatial taxonomy is restricted to `filled`, `empty`, and `mixed`, in a world where almost everything is `mixed`. `Mixed` cells are subdivided until an `empty`

region is found. However, it is hard to propagate this useful information to guide the search through nearby, unrefined cells in the subdivision. One goal of the algorithms and representations in this thesis has been to exploit *coherence* in the configuration space. The intuitive appeal is that the intersections of C-surfaces "go somewhere useful" (i.e., around the obstacles). We will adopt an approach which exploits the coherence of *C-Space* obstacles by moving along the intersections of high-dimensional C-manifolds[2] parallel to the obstacle boundaries.

Lozano-Pérez (1981) has described approximate solutions for Cartesian manipulators with 6 degrees of freedom (in principle) which consider 3-dimensional *slice-projections* of Configuration space. In practice these approximations are only reasonably accurate for Cartesian manipulators with 4 degrees of freedom. In principle, the *C-Space* constraints on motion defined by Lozano-Pérez (1983) can be extended directly to a 6 degree of freedom planner; indeed, this is our starting point. However there are many interesting and complex problems to work out (see Brooks (1983b) for another discussion of these problems). In particular, there are many unresolved mathematical details for the 6 degree of freedom case. Given the mathematical model, there still remains the issue of a complete planner which exploits the model.

1.5.4. Global Methods

Global methods for path planning attempt to construct a model of the connectivity of free-space which can be related to the *Voronoi* diagram (see Drysdale (1983)). In particular, Brooks (1983a) has implemented a 2-dimensional path-planner which models the free-space as an overlapping union of Generalized Cones (Binford (1971)). Each cone provides orientation constraints on motion within the cone, and these constraints are intersected to find a translational path along the cone axes (called *spines*) interspersed with rotations at the spine intersections. This work was extended to a six-link manipulator for moving payloads with 4 degrees of freedom (Brooks (1983b)). The extended algorithm uses the same cone model, but sweeps each cone vertically to build prisms at horizontal slices through the workspace. This method works well when the payload (or polygon) is small and

[2] A C-manifold is a manifold in a configuration space.

convex in a relatively uncluttered obstacle environment. It is not at all clear how to extend the algorithm to large, non-convex moving objects, or how to consider more than one rotational degree of freedom at a time. Nevertheless the concept of computing "freeways," or "channels" through free-space is attractive in that it can provide global guidance to local algorithms (such as *C-Space* methods), and can enumerate good hypotheses about candidate paths through complex workspaces.

Using an approach called *retraction*, Ó'Dúnlaing and Yap (1982), Ó'Dúnlaing, Sharir and Yap (1982) construct a Voronoi diagram for a two-dimensional workspace and consider moving simple objects (a disc, a line-segment) along it. This technique was mentioned by Brooks (1983a). It has not yet been extended to polygonal objects or 3-dimensional cases. We will address this issue by considering Generalized Voronoi Manifolds.

1.5.5. Approximation and Completeness

Planning problems have two components: characterizing the constraints, and searching for a solution which satisfies the constraints. One attempts to achieve a *complete* (in some sense, "exact") characterization of the constraints, and a complete search algorithm for the representation. Since the Mover's problem is a continuous mathematical decision problem, we must in general consider a *discretized* version of the problem (see Reif (1979)), for example, we might represent the polyhedral input models as systems of linear inequalities within a fixed accuracy ϵ, with $0 < \epsilon < 1$. In fact, there are two kinds of resolution limit. Any algorithm which employs real arithmetic has a resolution limited to the machine precision. (Schwartz and Sharir (1982a) employ rational and algebraic numbers instead). For the find-path problem, we are interested exclusively in the physically realizable paths, that is, those paths lying entirely within open sets of free space. The resolution limit Reif mentions is essentially a bound on how small an open set can become before it is no longer considered open. The open set resolution limit is typically greater than the machine precision.

Almost all find-path search algorithms are complete only to this fixed resolution; the notable exception is Schwartz and Sharir (1982a), which appears to be search-complete and resolution independent. We should stress that for a complete

representation, the resolution-dependence is in practice not a severe restriction. However, the effect of a complete search algorithm running on an approximate characterization of the constraints is not clear. In principle, in case of search failure, it is sometimes possible to refine the approximation and redo the search until a path is found. This possibility has rarely been exploited however, and introduces a number of unpleasant technical and conceptual issues. A complete search running on an approximate representation will in general result in an incomplete algorithm. For these reasons we would prefer a complete characterization of the constraints coupled with a complete search algorithm.

We will place this thesis in the context of previous work by considering the following criteria:

(i) For what degrees of freedom does the algorithm apply?

(ii) Is the representation (the characterization of the constraints) complete?

(iii) Is the search complete (at a given resolution)?

(iv) Has the algorithm been implemented?

Approximate Representations

Much of previous work has focused on approximate characterizations of the constraints. Approximate representations may (1) artificially restrict the degrees of freedom in a problem, (2) bound objects in real-space by simple objects such as spheres, or prisms with parallel axes, while considering some subset of the available degrees of freedom, (3) discretize configuration space at certain orientations, or (4) approximate swept volumes for objects over a range of orientations. Such restricted planning systems may lose solutions which require exploiting all six degrees of freedom. An approximation of the obstacle environment, robot model, or *C-Space* obstacles can result in a transformed find-path problem which has no solution.

Some approximate algorithms -for example, those of Brooks-run quite fast for the class of problems that they address. In general, speed has been a motivating factor in the design of these approaches. We also observe that some approximate methods were motivated by the difficulties of modeling constraints in a full 6-dimensional *C-Space*. These difficulties in turn stemmed from unresolved mathematical problems relating to both *C-Space* itself and to the structure of

C-Space constraints. However, even with a complete mathematical model in hand, we are still confronted with the problem of devising a complete planner which works using the full set of constraints.

The configuration space of the three dimensional classical Movers' problem with six degrees of freedom is $\Re^3 \times SO(3)$, where $SO(3)$ denotes the three-dimensional rotation group. In this thesis we first complete the mathematical framework for the configuration space $\Re^3 \times SO(3)$ and present solutions for some heretofore unsolved problems. This foundation then allows us to propose and construct a complete planner exploiting the full set of constraints and 6 degrees of freedom for motion planning in $\Re^3 \times SO(3)$.

In this section, we characterize the completeness of previous work. Unless noted, search-completeness is resolution-dependent. Schwartz and Sharir (1982a) describe complete representations and complete (unimplemented) search algorithms for 2D and 3D. These theoretical algorithms appear to be resolution-independent. Brooks and Lozano-Peréz (1983) describe complete representations and search algorithms for the problem in $\Re^2 \times S^1$. Lozano-Pérez (1981, 1983), Lozano-Pérez and Wesley (1979) give approximate representations (except for translation) with complete search algorithms for $\Re^2 \times S^1$ and $\Re^3 \times SO(3)$. These approximate representations also model Cartesian manipulators. Most algorithms for $\Re^3 \times SO(3)$ can be extended for Cartesian manipulators in a similar manner. For translations, Lozano-Pérez' algorithms are complete to the machine resolution. Brooks (1983a) provides an approximate constraint characterization with a complete search algorithm for $\Re^2 \times S^1$; Brooks (1983b) extends this for a linked arm carrying a payload with degrees of freedom $\Re^3 \times S^1$. A significant contribution of Brooks was the addressing of the issue of jointed arms. The open set resolution limit for the Voronoi methods (for simple objects in two dimensions) is no larger than the machine precision. Udupa (1977) and Widdoes (1974) used approximate representations and incomplete search algorithms in addressing find-path for jointed arms.

In this light, we can characterize our algorithm as follows:

- *This thesis presents the first implemented, representation-complete, search-complete algorithm (at a given resolution) for the classical Movers' problem in*

$\Re^3 \times SO(3)$.

1.6. An Outline of this Thesis: Research Contributions

In this thesis we present a local algorithm for the six degree of freedom classical Movers' problem. The channel based algorithm developed in Donald (1984) is described in Donald (1983a).

At the heart of this research lie certain mathematical developments that may seem fairly abstract at first reading. To motivate the mathematics, we first present, in chapter 2, the design and implementation of a six degree of freedom planning system for the classical Movers' problem. The description of the planning algorithm assumes that certain representations and mathematical tools are available. In subsequent chapters, we develop these tools in answer to the following questions, for which chapter 2 assumes solutions:

Representational and Algorithmic Questions

(i) What is an appropriate representation for constraints in a six dimensional *C-Space*? (Chapter 3).

(ii) In the six dimensional *C-Space* of the classical Movers' problem, the domain of each constraint is the product space of $\Re^3$ and a complicated three-dimensional manifold (with boundary) on the projective three-sphere. What are these regions, and what is their structure? What representation can be used for these domains? (Chapter 3).

(iii) How do we plan motions using constraints whose domains change with the motions? (Chapter 5).

(iv) Given a trajectory in *C-Space*, it is necessary to find where it intersects the boundary of *C-Space* obstacles. How can trajectories be intersected with C-surfaces whose domains change along the trajectory? (Chapter 4, 5).

(v) How can intersection manifolds be constructed in *C-Space*? (Chapter 4).

(vi) How are motions planned that slide along C-surfaces and intersection manifolds? (Chapter 4, 2).

(vii) How can rotation space be decomposed into regions where the set of applicable constraints is invariant? (Chapter 5).

How to Read this Thesis

Chapter 2 –covering the design and implementation of the search algorithm in *C-Space*—presents the most heuristic component of this research. It is also in some sense the most accessible chapter to the non-specialist. However, do not confuse chapter 2's implementation details and search heuristics with the representational and algorithmic framework developed under the considerably more formal ægis of chapters 3 through 6. The thesis is structured so that those prefering a presentation more in keeping with the traditional style of mathematical exposition may read the chapters on 6DOF planning in the alternative order:

(1) Geometric Planning Problems

(3) Questions of Representation: C-functions and Applicability Constraints in a Six Dimensional Configuration Space

(4) Mathematical Tools for Motion Planning in a Six Dimensional Configuration Space

(5) Moving Through Rotation Space

(6) The C-Voronoi Diagram and its Relationship to Intersection Manifolds

(2) A Planning System for the Classical Movers' Problem with Six Degrees of Freedom.

In the alternative order, the representations and algorithms are derived and presented first, and the application and implementation is presented last.

Chapter 3 presents a formal framework in which several open questions about configuration space constraints—notably (ii) (above) - may be solved. Chapter 3 also derives fundamental structural properties of *C-Space* constraints, in particular, the domains and domain topology of C-functions for the classical Movers' problem. We call these domains *applicability constraints*. Chapter 4 addresses the *intersection problem* in high-dimensional *C-Space*: how to construct and slide along intersection manifolds, and how to intersect trajectories with C-surfaces and applicability constraints. We demonstrate the form of the intersection manifolds for $\Re^3 \times SO(3)$ and $\Re^2 \times S^1$. Chapter 5 discusses algorithms for moving through rotation space, and for decomposing rotation space into equivalence classes where the set of applicable constraints is invariant. In chapter 6, we extend the concept of the generalized Voronoi diagram (which Drysdale (1983) defined for the plane) to the six

dimensional *C-Space* $\Re^3 \times SO(3)$, to provide a formal and constructive definition of the *C-Voronoi Diagram*, or *CVD*. The CVD is an attractive construction, in that it contains a representative component for each "branch" of free space. Each such component is a submanifold of dimension $0 \leq d \leq 5$, called a *Voronoi manifold*. We will derive the following connection between intersection manifolds and the CVD:

Theorem: (*The Equivalence Theorem for intersection manifolds and the CVD*). Let p be a path along the CVD. p lies along a connected chain of Voronoi manifolds, $V_1, \ldots, V_k$. We demonstrate that for each Voronoi manifold V_i, there exists an equivalent intersection manifold of level C-surfaces, I_i. Furthermore, we also show that for every connected chain of Voronoi manifolds, there is an equivalent connected chain of intersection manifolds (of level C-surfaces). (The equivalence we demonstrate is actually stronger than homotopic equivalence, but the additional details are too complicated for this chapter).

2
A Planning System for the Classical Movers' Problem with Six Degrees of Freedom

In this chapter, we describe the design and implementation of a planning system for the classical Movers' problem with six degrees of freedom. The planning algorithm required the solution of the seven "Representational and Algorithmic Questions" listed at the end of chapter 1. The solutions to these problems are presented in subsequent chapters.

In this chapter we will simply assume that these problems are solved, and proceed to employ the solutions in constructing a planning algorithm. Of particular importance will be two effective procedures, which address the *intersection problem* in *C-Space*:

(I) Given two or more level C-surfaces, construct their intersection manifold. (Chapters 3 and 4).

(II) Given a C-surface and a trajectory, find their intersection. Determine whether the intersection lies on the boundary of a *C-Space* obstacle. (Chapters 4 and 5).

The immediate application of (I) is the *sliding* problem: How to slide along one level C-surface, and how to slide along the intersection of two or more level C-surfaces.

Using the point navigation operators (chapter 1), we implemented a best-first search algorithm in *C-Space*. The algorithm has nice theoretical properties which include completeness (at a resolution). This chapter describes the heuristic search, with particular emphasis on the heuristic strategies that evaluate local geometric information, and on the interaction of these strategies.

2.1. Definitions

A topological space M is called an n-dimensional *manifold* if it is locally homeomorphic to $\Re^n$. A *chart* is a way of placing a coordinate system on M: if U and V are open subsets of M, two homeomorphisms $f : U \rightarrow f(U) \subseteq \Re^n$ and $g : V \rightarrow g(V) \subseteq \Re^n$ have C^∞ *overlap* if the maps

$$f \circ g^{-1} : g(U \bigcap V) \rightarrow f(U \bigcap V)$$
$$g \circ f^{-1} : f(U \bigcap V) \rightarrow g(U \bigcap V)$$

are also C^∞ (that is, possessing continuous partial derivatives of all orders). A family of pairwise C^∞-overlapping homeomorphisms whose domain covers M is called an *atlas* for M. A particular member (f, U) of an atlas $\mathcal{U}$ is called a *chart* (for the atlas $\mathcal{U}$), or a coordinate system for U. For a good introduction to differential geometry, see, for example, (Spivak, 1979).

In this thesis we usually specify charts via the inverse form $h : R \rightarrow M$ (where R is an open subset of $\Re^n$) with the understanding that it is the inverse (or set of local inverses) h^{-1} which provides the family of charts $\{(h^{-1}, W_i)\}$, for $\bigcup_i W_i = h(R)$. As an example, consider the map h that specifies a chart for a five dimensional level C-surface:

$$h : \Re^5 \rightarrow \Re^3 \times SO(3)$$
$$(y, z, \psi, \theta, \phi) \mapsto (-\frac{E_2 y + E_3 z + E_4 - \ell}{E_1}, y, z, \psi, \theta, \phi).$$

Here the E_i are smooth, real-valued functions on $SO(3)$, that is, $E_i : (\psi, \theta, \phi) \rightarrow \Re$. The inverse map h^{-1} is obvious, and provides a chart for the five dimensional submanifold of $\Re^3 \times SO(3)$. In subsequent chapters we will derive such charts, in the form of h; in this chapter, we will take them for granted.

2.2. Introduction

We are now ready to describe a planning system for the find-path problem in $\Re^3 \times SO(3)$. The algorithm has the structure of a search and is complete (for a given resolution). The basic idea is as follows: we are able to define and implement certain *local operators*. When applied at a configuration in *C-Space*, a local operator attempts to move the robot in a specified direction until either the subgoal or an intervening C-surface is reached. The local operators have the general form

$$Move(X{:}configuration, \hat{v}{:}direction, limit{:}configuration),$$

and are designed to return X', the configuration reached in direction $\hat{v}$, and the reason for stopping (which will either be "reached subgoal" or the name of the C-surface which halted progress). The local operator assumes that X is in free-space, and ensures that there exists a collision-free path along $\hat{v}$ taking the robot from configuration X to X'. Furthermore, we insist that $limit = X + t\hat{v}$, for some positive t. In general, $\hat{v}$ can be represented as a tangent vector to $\Re^3 \times SO(3)$; the space of directions is clearly locally homeomorphic to $\Re^6$.

Many different *Move* operators can be defined. Let $X = (x, \Theta)$. We will restrict $\hat{v}$ to be either a pure translation

$$\hat{v} \in \Re^3 \times \{\Theta\}$$

or a pure rotation

$$\hat{v} \in \{+\hat{\psi}, -\hat{\psi}, +\hat{\theta}, -\hat{\theta}, +\hat{\phi}, -\hat{\phi}\}.$$

The closure of these operators is complete for the space of configurations. By this we mean that in the absence of obstacles, there is some finite sequence of operators which carries any configuration X into any other configuration Y. It is often convenient to think of these operators as $Translate(X, \hat{u}, x')$ (where $\hat{u} \in \Re^3$ and x' is a goal translation) and $Rotate(X, \hat{\varphi}, \varphi')$ (where $\hat{\varphi}$ is an angular direction

and φ' is a goal angle). The theory and implementation of *Translate* and *Rotate* is discussed in chapters 3, 4, and 5.

Given the local operators, we can define more sophisticated local strategies for spatial reasoning. These strategies are implemented by *local experts*[1] in *C-Space*. For example, one local expert attempts to circumnavigate *C-Space* obstacles by sliding along intersections of level C-surfaces. Another, "greedy" expert tries to translate or rotate straight towards the goal. A local expert typically examines the local geometric environment of C-surfaces, their normals and intersections. It also takes into account the history of planning. The local experts can be thought of as issuing "commands" in terms of the local operators. Depending on the results of these attempted motions, an expert may issue other local operator commands, and either directly invoke or leave a forwarding message for another local expert.

To summarize: a local *operator* is an algorithm for moving along a specific trajectory until a constraint is encountered (or a subgoal is reached) . A local *expert* is a strategy for choosing the trajectory based on an examination of the history of planning and the local geometry. When a local expert chooses a trajectory, it calls on some sequence of local operators to realize it.

[1]The term *local expert* was brought to my attention in discussions with Van-Duc Nguyen (Nguyen (1983)), Tomás Lozano-Pérez, and Rodney Brooks.

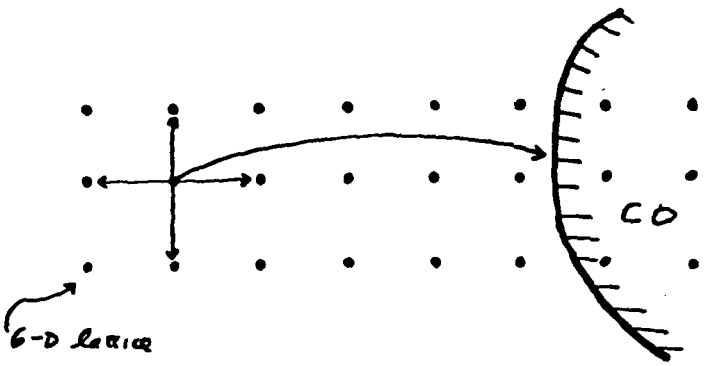


Figure 2.1. Schematic illustration of the "Bumble" strategy (an exhaustive search). A fine six-dimensional lattice is thrown across *C-Space*. By exploring from one configuration to its neighbors in the lattice, a path will eventually be found, if one exists at the lattice resolution. Fortunately, it is also possible to take large steps in the lattice, and simply record the neighborhoods the path visits.

2.2.1. Planning and Search

The planning algorithm is implemented as a search of configuration space. The search constructs a graph of neighborhoods which have been explored. (We will be more precise about the term *neighborhood* later). Each node in the search graph is associated with a configuration and contains information abou' the local geometry and the history of planning. The search algorithm chooses a node for exploration. Several local experts are then applied at that node. Each expert can produce a new search node. All of these are sons of the explored node, and are added to the search queue. The new sons are connected to their father by the arcs of the search graph and each son may be thought of as an *exploration* from the father.

If at any point in the search, two explorations reach the same neighborhood,

the planner attempts to merge the associated nodes into one node.

The search algorithm is Best-First (Nilsson (1980)) with the metric of progress established as distance from the goal. (This requires placing a metric on both translation and rotation space). Other search measures (such as path length, or time) would also be possible, and an A^* search strategy could be exploited to find optimal paths. In practice this would probably require adding new local experts in order to ensure reasonable performance.

As search nodes are explored, they are entered in a priority queue, called the *search queue*. The nodes in the search queue are ordered by the search metric. Some search strategies we discuss require two search queues: when the primary queue is exhausted, then nodes from the reserve queue are explored.

We will proceed as follows. First, using the local operators alone, we can define a complete search strategy (at a given resolution). This search strategy can be considered the most primitive local expert, and is known as the "Bumble Strategy." By applying the Bumble strategy at every search node, we are guaranteed to find a path (at a given resolution) if one exists.

Next, we will define more complicated local experts which will be applied to search nodes at the same time as the Bumble expert. These experts greatly improve the performance of the planner.

2.3. A Complete Search Strategy

A search node is associated with a configuration. Every configuration is in turn associated with a neighborhood of *C-Space*. The neighborhoods form a partition of *C-Space*. Since many configurations are associated with one neighborhood, so several search nodes may have configurations lying in the same neighborhood.

Assume the neighborhoods are "small." If the configurations of two search nodes are in the same neighborhood, it indicates that they should, if possible, be merged into one node, since they are close together. By keeping track of the set of explored neighborhoods, we can avoid redundant explorations. If the neighborhoods are sufficiently small, then the search will be complete at a resolution closely related to the neighborhood size.

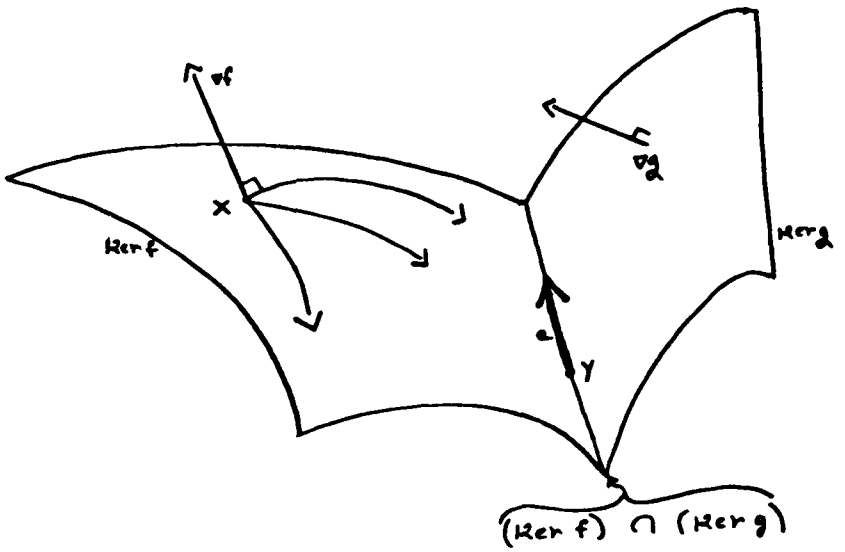


Figure 2.2. $f, g : \Re^3 \times SO(3) \rightarrow \Re$ are C-functions which describe two level C-surfaces, $\ker f$ and $\ker g$. The level C-surfaces are smooth, 5-dimensional manifolds parallel to *C-Space* obstacle boundaries. From $X \in \ker f$, three paths sliding along the level C-surface $\ker f$ are shown. Each path is orthogonal to ∇f. The *sliding expert* plans such paths along 5-D level C-surfaces. $(\ker f) \bigcap (\ker g)$ is the intersection of the two level C-surfaces, and is a 4-dimensional manifold. The *intersection expert* plans paths along intersection manifolds. Such a path p is shown from configuration Y.

It is possible to devise a complete search strategy (at a given resolution) using just the local operators. We first throw a fine six-dimensional lattice[2] over configuration space. The lattice is used to keep track of the *state* of the planner, i.e., which neighborhoods have been explored, and for computing the *connectivity* of these neighborhoods. The lattice will "wrap around" in the rotational dimensions, but this is easily implemented using modular arithmetic. We will define an adjacency function for points in the lattice; in addition, when a neighborhood is explored, the corresponding node in the lattice is marked. When a search node is chosen for exploration,

[2]I.e., the factor spaces of the parameter space are quantized, and the lattice is a partial order on the Cartesian product of the factor space quantizations.

(i) X, the configuration of the search node is mapped to L, a point in the lattice. L is the *name* of the neighborhood $\mathcal{N}(L)$ centered on L, which contains X.

(ii) The unexplored neighborhoods adjacent to $\mathcal{N}(L)$ are found. Each of these neighborhoods is also identified by a central lattice point.

(iii) The planner attempts to move to each of the unexplored, adjacent neighborhoods.

(i) has the effect of mapping a neighborhood of *C-Space* to a canonical element (which lies on the lattice) in its interior. These neighborhoods decompose $\Re^3 \times SO(3)$ into equivalence classes with the same canonical element. When a neighborhood is reached for the first time, we mark its lattice point as explored. The search terminates when a neighborhood containing the goal is reached, and when that exploration can be connected to the goal configuration.

2.3.1. Implementation of Neighborhoods and Lattices

In principle, it is possible to implement the lattice as a six-dimensional array (with modular indexing for the rotational dimensions). In practice, for any fine resolution, this array will be enormous, and very sparse. Although an adversary can design a find-path problem for which our planner must explore the entire lattice, in practice this does not occur. However, we must maintain a record of what neighborhoods have been explored, in order to generate the unexplored neighbors for a search node. Since the array is sparse, we will employ a different strategy.

A partial order can be defined on lattice points by considering them as six-dimensional vectors. This order has no particular geometric significance for the rotational dimensions, but it can be used to store explored lattice points in a binary tree. Since the vast majority of neighborhoods are never explored, the tree it typically small, even for fine lattices. To mark a lattice point as explored, we insert into the binary tree. To find whether a lattice point has been explored, we search the tree.

It is desirable to employ a fine lattice in order to ensure completeness at a fine resolution. The use of a binary tree to record explored configurations effectively removes the problem of lattice size for storing explored configurations. For example, if we segment *C-Space* into an $N \times N \times \ldots \times N$ lattice, then an array would have to be N^6 long. But the binary tree need store only the explored locations, and (if height-balanced) can access any leaf in $O(\log N)$ operations.

If the lattice resolution is fine, then the planner as described so far will take very small steps for each search exploration. This has been remedied as follows: If a local operator is invoked to find whether *limit* may be attained from X in direction $\hat{v}$, it must effectively intersect a path in direction $\hat{v}$ with all C-surfaces. It is not much harder to find the *first* constraint along the path $p(t) = X + t\hat{v}$ (even if it is beyond *limit*): in particular, we note that all intersections along the path p may be sorted on distance from X. The complexity of finding this first intersection along p is independent of the lattice resolution (since the intersection algorithm has nothing do do with the lattice; see chapter 5). We can "sample" the portion of the path which lies in free space at the lattice resolution. All of these configurations are then marked as "explored", and as reachable from their immediate neighbors along the path. Thus they form a connected chain in the lattice along the path p. While all these configurations are in some sense sons of X, in practice we will select only one or two to be entered in the primary search queue. These sons might be (1) the son which is closest to the goal, and (2) some son at a reasonably large step away from X. This step size, called the *Bumble resolution*, might be 3 to 10 times the lattice resolution. The other sons should be kept on a reserve queue, which can be explored when the primary search queue is depleted or exhausted.

In practice, it may preferable to enter *ranges* in the exploration tree, for example, to record that all lattice points

$$(x, y, z, \psi, \theta, \phi) \leq L \leq (x + kd_T, y, z, \psi, \theta, \phi)$$

(for some integer k) are explored. This requires keeping an exploration tree of *lines* instead of configurations, with the intent of minimizing the number of exploration tree entries. When lines are entered into the tree, they may be merged with previous lines to form connected components of explored regions. These operations are supported by hierarchical subdivision algorithms. At this point in the experimental use of the planner, it is still too early to tell whether this optimization is necessary.

In practice we have had no problem in selecting a very fine resolution for the lattice (one selects a fine lattice resolution, and a considerably larger Bumble resolution or step size, as described above). This lattice-based strategy is not only theoretically complete for a given resolution, but has also been used to find very complicated paths for the 6 degree of freedom classical Mover's problem. However, the algorithm has an "excessively local" flavor—it is clumsy and quite slow when employed alone (hence the strategy's name). We can construct much "smarter" heuristic experts which attempt to exploit coherence in *C-Space*. When these experts are used in conjunction with the Bumble strategy, we obtain a planner which is not only complete, but which can solve complicated problems in a reasonable amount of time. We continue to find the lattice useful for recording the planner's explorations by the local experts.

2.3.2. Keeping Track of Connectivity

Suppose a subsequent exploration reaches the same neighborhood. There are two choices, which we call the *mark* algorithm and the *connect* algorithm:

The Mark Algorithm. Discard the exploration, since the neighborhood is already explored. In practice, the mark algorithm often suffices for path-finding. The mark algorithm computes a directed, spanning tree T of explored neighborhoods, which is rooted at the start configuration.

The Connect Algorithm. Connect together the search nodes for all explorations to that neighborhood. The connect algorithm is more complicated, and requires the following bookkeeping (see figure 2.3). Let $\mathcal{N}$ be a neighborhood of $\Re^3 \times SO(3)$, and $L \in i(\mathcal{N})$ be a lattice point which is the canonical element for $\mathcal{N}$. Suppose X is an exploration of $\mathcal{N}$, i.e., $X \in \mathcal{N}$ is the final configuration in some motion reaching $\mathcal{N}$. Let $s(X)$ denote the search node for X. (If X is the first exploration of $\mathcal{N}$, then create a search node $s(L)$ for L). Determine whether there exists a path from X to L (using the local operators). If so, connect $s(X)$ and $s(L)$ together.

The connect algorithm computes a more complete connectivity graph for the neighborhoods of *C-Space*. It computes an undirected graph H of explored neighborhoods, which may contain cycles. As long as H is connected, then T

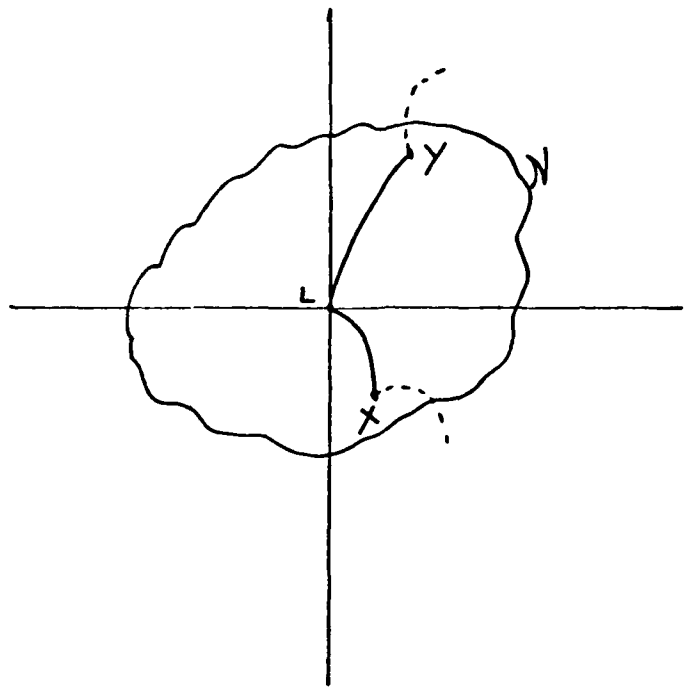


Figure 2.3. The lattice point L is at the center of a neighborhood $\mathcal{N}$ of *C-Space*. Search explorations arrive at configurations X and Y in $\mathcal{N}$. The planner attempts to find a path connecting X and Y, by trying to connect both configurations to L.

is a spanning tree for H, and the mark algorithm is complete for planning a connected path along H. However, not all planning strategies admit this kind of "connected planning." In particular, when we consider strategies which construct partial paths and planning islands (which may later connect up), the connect algorithm is necessary. (See the *Suggestor* strategy, below, for an example).

2.3.3. Discussion of the Bumble Strategy

Suppose the lattice spacing is d_T and d_R in the translational and rotational dimensions. Then the adjacent lattice points to $L = (x, y, z, \psi, \theta, \phi)$ will be:

$$(x \pm d_T, y, z, \psi, \theta, \phi)$$
$$(x, y \pm d_T, z, \psi, \theta, \phi)$$
$$(x, y, z \pm d_T, \psi, \theta, \phi)$$
$$(x, y, z, \psi \pm d_R \pmod{2\pi}, \theta, \phi)$$
$$(x, y, z, \psi, \theta \pm d_R \pmod{2\pi}, \phi)$$
$$(x, y, z, \psi, \theta, \phi \pm d_R \pmod{2\pi})$$

Each adjacent lattice point is the center of a neighborhood of configurations which is contiguous to the neighborhood of L. Each such neighborhood can be reached (if it is in free space and there is no intervening C-surface) by the local operators *Translate* and *Rotate*. Since there are 12 neighbors for each lattice point, we have found it inadvisable to explore them all for each search node expansion. Instead, the set of unexplored adjacent neighborhoods is ranked (in terms of proximity to the goal), and motions towards the top k_T translational and k_R rotational neighbors are attempted. (Typically, $k_T \approx 3$ and $k_R \approx 2$). If the node is reexplored later, motions toward $k_T + k_R$ more of the unexplored neighbors will be attempted (if there are that many left). When using the mark algorithm (above), we say an exploration is successful if it reaches a new (unexplored) neighborhood. If an exploration is successful, then a new search node is created and the neighborhood is marked as explored. Since the neighborhood's "name" is its lattice point, this simply corresponds to marking the lattice point. Whether successful or not, all explorations are recorded at the parent search node so that they will not be tried again.

Suppose X is a configuration in neighborhood $\mathcal{N}(L)$, with associated lattice point L. The unexplored adjacent lattice points to L indicate a set of subgoals to be attained from X. The Bumble strategy ranks these subgoals, chooses some of them, and selects trajectories which may attain them. The local operators are then employed to (try to) realize the selected trajectories. These explorations are then recorded so that only new explorations will be pursued in the future. Note that the planner is not constrained to move along the lattice, and that although the subgoals lie on the lattice, the motion from X to any subgoal does not, unless $X = L$.

The local experts are considerably more sophisticated than the Bumble strategy.

Their subgoals need not lie on the lattice, and the motions specified to the local operators need not lie along the lattice. The lattice is still employed to keep track of the planning history and the connectivity of explored neighborhoods.

Clearly, the arcwise-connected sets of lattice points are closed under the operators *Translate* and *Rotate*. If a path exists at the lattice resolution, then the search is guaranteed to find it. We see now exactly what the *resolution* for this find-path algorithm is: by choosing a sufficiently fine lattice, the algorithm is (trivially) complete at the lattice resolution. As we saw above, we can choose a very fine lattice with little computational overhead. One final point: the start and goal configurations may not lie directly on the lattice. This is not a problem, however, since the local operators can ensure that there exists a path from the start and goal to the nearest lattice point.

2.4. Local Experts for the Find-Path Problem

2.4.1. Path Planning versus Continuous Intersection Detection: Why We Need Local Experts

The *Translate* and *Rotate* operators detect collisions along continuous trajectories.[3] Given these operators, it is possible to devise a complete path-planning algorithm based on something like the Bumble strategy, above. However, while complete, this is not a particularly *good* algorithm, in that it says nothing about how or when the operators should be applied. The domain of the operators is large and for realistic path planning, it is necessary to know where, and in what directions to apply them.

> Algorithms which can detect intersections with obstacles for a robot following a continuous trajectory say nothing about how to plan these trajectories. However, they can be used to find a path by exhaustive search.

The *Translate* and *Rotate* operators use the constraints in *C-Space* to detect collisions. However, these constraints can also be employed to plan paths. In chapter 1, we proposed an idealized planner which constructed the intersection

[3]This discussion also holds for the general *Move* operator.

manifolds of level C-surfaces, and slid along these manifolds to navigate around *C-Space* obstacles. Such a planner could exploit *coherence* in configuration space: by examining *C-Space* constraints an algorithm can be devised for intersecting and sliding on C-surfaces to circumnavigate *C-Space* obstacles. In the following sections, we describe a planner which approaches the idealized planning algorithm of chapter 1. The local experts are strategies for reasoning about the local geometry of configuration space, and for exploiting geometric constraints to plan collision-free paths. When applied to a search node, each local expert examines the local geometry and history of planning to propose one or more path segments. Each path segment is realized by means of the local operators, which ensure that a collision free path exists.

2.4.2. Designing Local Experts

In the exploration tree of *C-Space* neighborhoods, we have seen one type of information that must be maintained for planning. In designing local experts, we must address the following questions:

(i) What constitutes a local description of a (level) C-surface?

(ii) What information should be stored at a search node?

(i) can be stated, "What constitutes a sufficiently rich description of the local geometry in *C-Space* to allow robust local experts?" (ii) relates more to the history of planning, and the connectivity of the explored search neighborhoods. For example, we want to record the results of previous applications of experts at a search node, and the adjacent nodes in the search graph.

The Local Description of a C-surface

A C-surface has a normal at point X. Motions tangent to the C-surface at X will have instantaneous velocities orthogonal to the normal. We must characterize the normal and tangents to a C-surface in order to plan trajectories which slide along it.

Let f be an applicable, positive-valued C-function at X. We can check that f is non-redundant at X (see chapter 6); alternatively, we may heuristically assume f is non-redundant if its value at X is small. We wish to develop a local characterization

of f at X, that is, of the level C-surface $S = \{ Y \mid f(Y) = f(X) \}$ about X. We should think of S as the kernel of the auxiliary function

$$f_X : \Re^3 \times SO(3) \rightarrow \Re$$
$$Y \mapsto f(Y) - f(X).$$

The local characterization will have two parts, one of which is invariant, and one of which will change for different subgoals. The invariant part of the description is a pair,

$$\Big(f(X), \nabla f \Big)$$

consisting of the *value* of f at X and the *normal* to S at X. Now, since $\Re^3 \times SO(3)$ is not a vector space, the normal $\nabla f(X)$ to S at X will depend on the Riemannian metric defined on the tangent space at X. We will employ a metric which admits construction of $\nabla f(X)$ using the partial derivatives of f at X, with respect to the parameterization of *C-Space*. Hence if rotations are parameterized by Euler angles, then $\nabla f = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}, \frac{\partial f}{\partial \psi}, \frac{\partial f}{\partial \theta}, \frac{\partial f}{\partial \phi})$.

Assume that ∇f is normalized to be a unit vector. We now wish to characterize the relationship of the C-surface to some subgoal, G: this requires some way of talking about directions in $\Re^3 \times SO(3)$. Specifically, we wish define a "vector" algebra on configurations, such that

$$\lim_{G \rightarrow X} \|G - X\| = 0$$

and

$$\lim_{G' \rightarrow G} (G - X) \cdot (G' - X) = 1.$$

These equations express the vector space characteristics which are required for our computations on tangent vectors. To construct this algebra, it is possible to define a field of inner products over $\Re^3 \times SO(3)$, i.e., to define an inner product on the tangent space to each point. Thus $\Re^3 \times SO(3)$ is a Riemannian manifold (see Erdmann (1984)). If two tangent vectors-i.e., directions-are applied to the same point, this inner product allows us to talk about the angle between two such tangent vectors, or of the angle between an arbitrary tangent vector to $\Re^3 \times SO(3)$ and the

normal to a C-surface. However, the inner product is somewhat arbitrary for our application. Alternatively, we could also construct geodesics on P^3, the 3-sphere with antipodal points identified. These approaches are probably too elaborate for a heuristic strategy.

Heuristics for Evaluating Directions in the Tangent Space

A basic issue is that placing a metric on a non-abelian group, such as $SO(3)$, is a difficult problem. We will demonstrate the metric that our planner employs, and then show that it is adequate for this application. In particular, the metric is adequate when applied to three one-dimensional slices of $SO(3)$. (These are the slices considered by the *Rotate* operator). Note, however, that a metric may also be derived by representing rotations as unit quaternions. In this case, the metric is obtained by considering rotations as points on S^3 embedded in $\Re^4$ (Brou 1983).

Suppose we employ rotation matrices to represent rotations. (The implemented planner uses Euler angles). If we are willing to tolerate singularities in the representation, it is often convenient to identify a rotation matrix in $SO(3)$ with the vector of three angles, (ψ, θ, ϕ) which determine it. The angles (ψ, θ, ϕ) form a three dimensional *angle space*, Q^3. The rotation matrix corresponding to (ψ, θ, ϕ) is of course $\mathcal{R}(\psi, \theta, \phi)$. (The singularities induce an equivalence relation on Q^3, where two points in angle space are equal when the rotation matrices they determine are equal). Most of the time, the identification of $SO(3)$ with Q^3 does not lead to problems. However, when we wish to compute directions, and differences of configurations, it is necessary to distinguish between $SO(3)$ and Q^3.

We can state this more concisely as follows: $SO(3)$ is a three dimensional manifold. The mapping $\mathcal{R}$ from Euler angles to rotation matrices is a chart for $SO(3)$:

$$\mathcal{R} : Q^3 \rightarrow SO(3).$$

We typically describe a rotation $\mathcal{R}(\Theta) \in SO(3)$ by its chart coordinates $(\psi, \theta, \phi) = \Theta \in Q^3$. This makes it convenient to identify Θ with $\mathcal{R}(\Theta)$, so that in general, instead of dealing with the manifold directly, we will work with a chart for the

manifold. In this section alone, however, we must distinguish between the domain and image of $\mathcal{R}$.

We can compute a direction in $\Re^3 \times SO(3)$ by simply subtracting two configurations (of course the angles must be subtracted $\pmod{2\pi}$) to yield a six-dimensional direction vector. Using this arithmetic, the *goal direction* is denoted $G - X$. We will use the convention that the first three coordinates of $G - X$ arise from $\Re^3$, and the second three coordinates arise from Q^3.

Let $G = (G_x, G_\Theta)$ and $X = (X_x, X_\Theta)$. Since $G - X$ is clearly well defined when G and X differ only by a translation, assume that G and X differ only by a rotation. Assume further that rotations are represented by Euler angles. Note that, in general $G - X$ is not a rotation which carries the moving object at orientation G into the moving object at orientation X. However, $G - X$ does represent the difference in orientation, i.e., it specifies a displacement in *angle space* which will carry G into X. For example, if $G_\Theta = (45°, 50°, 90°)$ and $X_\Theta = (45°, 45°, 45°)$ then there are rotation matrices $\mathcal{R}(G_\Theta)$ and $\mathcal{R}(X_\Theta)$ corresponding to each of G_Θ and X_Θ. (We use degrees, not radians in this example, since the symbol π will soon be used for a projection map). Note that

$$\mathcal{R}(45°, 50°, 90°) \neq \mathcal{R}(45°, 45°, 45°)\mathcal{R}(0°, 5°, 45°),$$

where $\mathcal{R}\mathcal{R}'$ indicates composition of rotations. However, the path in angle space

$$\begin{aligned} p(t) &= X_\Theta + t(G_\Theta - X_\Theta) \\ &= (45°, 45°, 45°) + t(0°, 5°, 45°) \end{aligned}$$

(for $t \in [0, 1]$) will work, since it corresponds to the rotational path

$$\begin{aligned} \mathcal{R}(p(t)) &= \mathcal{R}(X_\Theta + t(G_\Theta - X_\Theta)) \\ &= \mathcal{R}((45°, 45°, 45°) + t(0°, 5°, 45°)). \end{aligned}$$

Considering configuration space as the product space of the translation space and the angle space, we see that $G - X$ is well defined. $G_\Theta - X_\Theta$ specifies a

direction and a distance to be traveled in angle space in order to carry X_Θ into G_Θ. Furthermore, along the path from X_Θ to G_Θ, the corresponding rotations specified by the angle space trajectory p are well defined. For all $G \in \Re^3 \times SO(3)$, we will treat the space of directions $G - X$ as the tangent space T_X to $\Re^3 \times SO(3)$ at X. Properly, T_X is the product space of the tangent space to $\Re^3$ at X_x, and the three dimensional angle space Q^3.

We now define a map from $T_X \times T_X$ to the plane, which will function in place of an inner product. First, define the natural projection maps from T_X onto its factor spaces:

$$\pi_{\Re^3} : T_X \to \Re^3$$
$$(G - X) \mapsto (G_x - X_x)$$

$$\pi_\Theta : T_X \to Q^3$$
$$(G - X) \mapsto (G_\Theta - X_\Theta).$$

Let $u \cdot v$ denote the standard inner product on $\Re^3$, for vectors u and v. If u and v are projections (under $\pi_{\Re^3}$) of direction vectors in T_X, we say that u and v are *translationally orthogonal* if $u \cdot v = 0$. Let $(q_1, q_2, q_3), (w_1, w_2, w_3) \in Q^3$. Assume the each pair of angles q_i and w_i (for $i = 1, 2, 3$) is normalized so that

$$|q_i - w_i| \leq 180°.$$

(Note that this normalization is critical). Now, define

$$n_Q\Big((q_1, q_2, q_3), (w_1, w_2, w_3)\Big) = q_1 w_1 + q_2 w_2 + q_3 w_3.$$

n_Q will function in place of an inner product on Q^3. We say that two rotational directions q and w are *rotationally orthogonal* if $n_Q(q, w) = 0$.

We may now define Φ_X, which will function in place of an inner product on T_X. First, let

$$D = G - X$$
$$D' = G' - X.$$

Assume that D_x, D'_x, D_Θ, and D'_Θ are all normalized to be length 1 (where the length of D_Θ is defined as $n_Q(D_\Theta, D_\Theta)^{\frac{1}{2}}$). Finally,

$$\Phi_X : T_X \times T_X \to \Re^2$$
$$\Big(D,\, D'\Big) \mapsto \Big(\pi_{\Re^3}(D) \cdot \pi_{\Re^3}(D'),\, n_Q\Big(\pi_\Theta(D),\, \pi_\Theta(D')\Big)\Big).$$

So Φ_X yields a pair consisting of the dot product of the translational components of the direction vectors, and the n_Q product of the rotational direction vectors. If $\Phi_X(D,\, D') = (0, 0)$, we say that D and D' are orthogonal directions in the tangent space T_X. Note that two directions are orthogonal if, and only if, their translational components are orthogonal and their rotational components are orthogonal.

This discussion extends naturally to other representations for rotations. For example, if spherical angles (Kane and Levinson (1978)) are used, then the difference in orientation *is* the rotation carrying G into X, that is, $G_\Theta - X_\Theta$ is a rotation carrying the moving object at orientation X_Θ into the moving object at orientation G_Θ. We should stress that the natural Riemannian inner product (Erdmann (1984)) could be used instead of Φ_X. This would complicate the representations employed in subsequent chapters. Φ_X and n_Q are heuristic measures on directions in T_X. We will later discuss why, for our purposes, they are good heuristic measures.

Evaluating Normals and Gradients to C-Surfaces

The local description of a C-surface relative to some subgoal is designed to address the following qualitative questions:
(i) Is the C-surface locally tangent or locally orthogonal to the goal direction?
(ii) Is the C-surface locally orthogonal to any rotational motion?

Recall that a level C-surface $\ker f$ is described by a real-valued C-function f. Assume that normals and tangent vectors are appropriately normalized. Question (i) may be resolved by examining

$$\Phi_X((G - X),\, \nabla f(X)). \tag{2.1}$$

When (2.1) approaches $(0, 0)$, we say that $\ker f$ is locally tangent to the goal direction. Note that (2.1) makes sense: f maps parameters of the form $(x, y, z, \psi, \theta, \phi)$ to real

numbers, and hence the gradient of f,

$$\left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}, \frac{\partial f}{\partial \psi}, \frac{\partial f}{\partial \theta}, \frac{\partial f}{\partial \phi}\right)$$

is clearly a direction in T_X.

We will also employ

$$\pi_{\Re^3}(G - X) \cdot \pi_{\Re^3}(\nabla f(X)), \tag{2.2a}$$

When (2.2a) approaches 0, we say that $\ker f$ is (locally) translationally tangent to the goal direction. Symmetrically, when (2.1) (resp. (2.2a)) approaches $(1, 1)$ (resp. 1), we say that $\ker f$ is locally orthogonal (resp. translationally orthogonal) to $G - X$. A similar calculation yields the rotationally tangent and orthogonal C-surfaces to the goal direction:

$$n_Q(\pi_\Theta(G - X), \pi_\Theta(\nabla f(X))). \tag{2.2b}$$

Why Φ_X and n_Q are Good Heuristic Measures

Suppose that the rotational direction is along one of the axes. (Let us say the direction is $\hat{\phi}$). To tell whether a C-surface is rotationally orthogonal (or tangent) to the $\hat{\phi}$ direction, we simply examine the magnitude of $\frac{\partial f}{\partial \phi}$, which can be obtained directly from $\nabla f(X)$. This is because

$$n_Q(\hat{\phi}, \pi_\Theta(\nabla(f(X)))) = n_Q((0, 0, 1), (\frac{\partial f}{\partial \psi}, \frac{\partial f}{\partial \theta}, \frac{\partial f}{\partial \phi})) = \frac{\partial f}{\partial \phi}.$$

In other words, the map n_Q need not be employed. Since the implemented *Rotate* operator moves along the rotational axes in directions

$$\hat{v} \in \{+\hat{\psi}, -\hat{\psi}, +\hat{\theta}, -\hat{\theta}, +\hat{\phi}, -\hat{\phi}\},$$

this is the most common – but not the only – test for rotationally orthogonal (or tangent) C-surfaces. This information is used by the rotation experts to choose rotational subgoals that move away from C-surfaces.

Description of a Search Node

The following information is stored at each search node. Lazy evaluation is implemented so that some of these objects (for example, the set of all applicable C-surfaces) may not be computed until they are required.

(i) The configuration X of the search node.

(ii) The lattice point for X, which is the unique identifier for the neighborhood about X.

(iii) The applicability set at X.

(iv) A, the set of non-redundant constraints at X, sorted on increasing value. The non-redundant constraints may be approximated by the applicable constraints having small positive (or zero) values at X.

(v) The parent node.

(vi) The *From-Direction* (The direction traversed from the parent node to this node).

(vii) The sons of this node. These include "unsuccessful" explorations which did not reach a sub-goal, or which reached a previously explored neighborhood.

(viii) The C-surfaces on which X lies which also bound *C-Space* obstacles, that is, all $f \in A$ such that $f(X) = 0$ and $\ker(f)$ bounds a *C-Space* obstacle at X.

(ix) An *Explanation* of how this node was reached. An explanation typically includes the *name* of the local expert that planned the move, and enough information to reconstruct the move. For example, the experts which slide along level C-surfaces leave an explanation containing the names of the constraints, their *levels* at the parent node, and the *parameterization* chosen for the intersection manifold.

Much of the information stored at a search node is used to record the history of the planning. An expert which planned the move to a search node s will not be applied again with the same parameters. As an example, consider the Intersection expert, which attempts to slide along intersection manifolds, and the Greedy expert, which attempts to move straight towards the goal. We discuss these experts in more detail in the next section. If applied to s, the C-surface intersection expert will not attempt to construct and slide along the same intersection manifold which led to s, unless it can slide in a different direction along the intersection manifold. By recording the From-Direction for a node, the planner can avoid repeating unfruitful explorations. In particular, different experts can advise motion in the same direction; thus a particular intersection manifold may point in the

same direction which was previously (or simultaneously) attempted by the Greedy expert. Whether successful or not, reëxploration in this direction may be avoided by examining the From-directions of the sons of s. An additional constraint is provided by the From-Direction of s itself: there is typically no point in exploring back in the direction we came from. The process of leaving information for some expert which may be applied in the future is known as "forwarding." As we shall see, the performance of one expert can provide strong hints as to what expert should be applied next.

The planner computes local descriptions for the C-surfaces in $\mathcal{A}$. Naturally, parts of these descriptions will change for different subgoals. The local characterizations of C-surfaces allow the planner to find the set of C-surfaces to which the goal-direction is tangent (or orthogonal) as described above. When a planning direction is chosen, these C-surfaces clearly provide strong constraints.

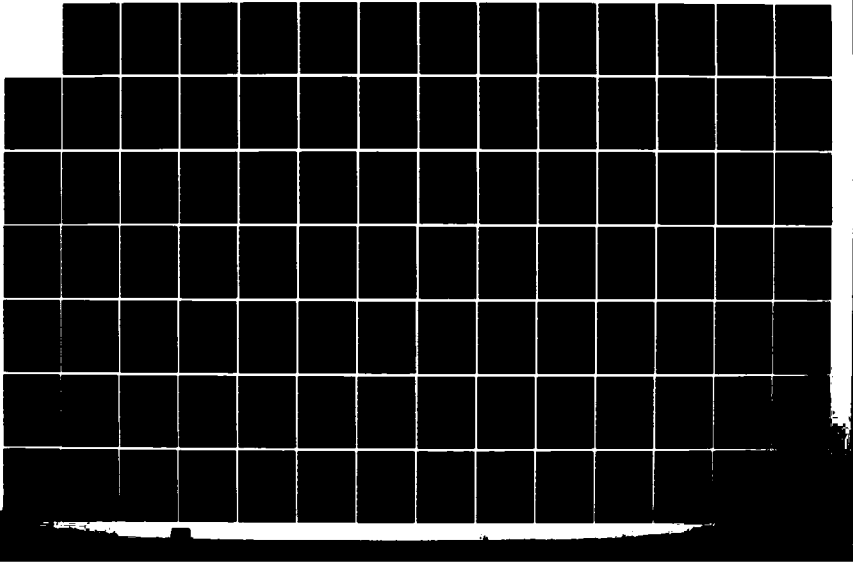
We are now ready to discuss the experts themselves. The Bumble strategy is also applied at each node, since it is a guarantee of completeness. In light of the previous discussion, we will omit any discussion of the detection and pruning out of explorations in unfruitful directions (as determined by the planning history). We will consider the application of particular experts to a search node s (at configuration X) which has parent s_0.

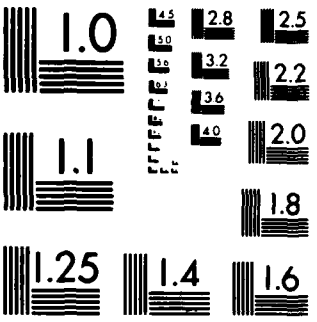
2.4.3. The Greedy Expert

The greedy expert attempts to translate or rotate directly towards the goal. The expert is necessary as an "end-game" strategy, in order to close in on a particular subgoal without worrying about finding the appropriate intersection manifold. The Greedy expert illustrates two important heuristics: *forwarding* and *backing off*. Suppose the greedy expert translates from a parent node s_0 to a son s. An appropriate explanation for the move will be left at s. If the same subgoal is intact when the planner explores s, the greedy expert will not attempt translation again. Instead, the rotation expert (see below) might be invoked. The effect is one of translating until an obstacle is hit, and then rotating to get around it. Alternatively, the sliding expert (which slides along level C-surfaces) might be invoked. This coupling of experts is termed the "hit and slide" strategy (see figure 2.4). However, the planner does not directly recurse by calling the sliding expert immediately after the greedy expert. Instead, a suggestion is left by way of explanation at s, and when s is explored in the search, the appropriate follow-up expert is invoked. The exact choice for which expert is invoked will depend on the history of planning (typically, what neighborhoods and directions have been explored from s_0 and s), and on the local geometry of C-surfaces about s.

Suppose that all experts moved the robot as far as they could, that is, moved until a constraint was hit and left the robot touching the constraint. This could result in jamming the robot up against many C-surfaces at once. It can prove very difficult to extricate the robot from this logjam situation. In fact, it is usually not preferable to move all the way up to an obstacle. Instead, we wish to detect this intersection with a planned trajectory p, and then *back off* from the obstacle boundary (along p). Thus if $p(0) = X$ and $p(1) = Y$ is the first intersection of p with *C-Space* obstacle boundary, then it makes good sense to move to $p(0.8)$. This has the effect of leaving the robot in the channel between obstacles instead of jamming it up in corners. Of course, if it is necessary to move to $p(.95)$ then the greedy and Bumble strategies will ultimately converge.

AD-A150 312 MOTION PLANNING WITH SIX DEGREES OF FREEDOM(U) 2/3
MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL
INTELLIGENCE LAB B R DONALD MAY 84 AI-TR-791
UNCLASSIFIED N00014-81-K-0494 F/G 6/4 NL





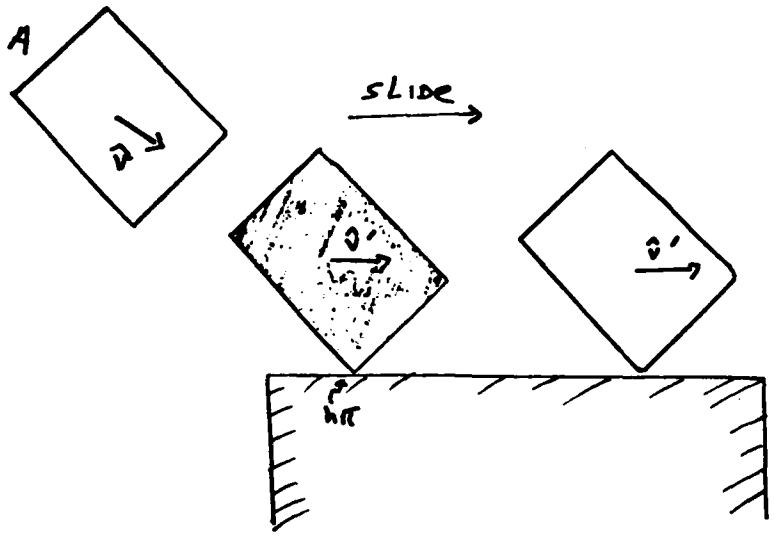


Figure 2.4. An idealized illustration of the hit and slide strategy. Some expert moves the robot in direction $\hat{v}$ until a C-surface S is hit at X. When the planner tries to move from X, the sliding expert is invoked to slide along S in the goal direction.

2.4.4. The Intersection Expert

The mathematics of intersection manifolds in $\Re^3 \times SO(3)$ is presented in chapters 3 and 4. The intersection expert attempts to find two C-surfaces in A whose intersection manifold contains a path which makes progress towards a subgoal. The path may be a pure translation or a pure rotation. We will begin by describing the process of finding a translational path which slides along an intersection manifold. First, all C-surfaces in A which are nearly translationally tangent to the goal-direction are selected. We select the first few of these which have the smallest value at X. Ideally, these are the closest non-redundant constraints at X. Call this set A'. The explanations for the moves from s_0 to s and from s to any sons of s will yield a set of previously explored intersection manifolds. (An intersection manifold may be identified by the name of the intersected C-surfaces, their levels, and the chosen parameterization). The C-surfaces in A' are pairwise intersected

(see chapter 4), after appropriate pruning as indicated by previously explored intersection manifolds. Each intersection manifold $(\ker f) \bigcap (\ker g)$ is constructed. A translation or rotation vector $\hat{v}_{f,g}$ is chosen such that the path $p_{f,g}(t) = X + t\hat{v}_{f,g}$ slides along the intersection manifold of the two level C-surfaces $\ker f$ and $\ker g$ at X. The intersection expert then selects the direction $\hat{v}_{f,g}$ which is closest to the goal direction (and which is not pruned out by consideration of the planning history). Suppose $\hat{v}_{f,g}$ is a pure translation. The local operator *Translate* is called to move from X in direction $\hat{v}_{f,g}$ until a C-surface is struck[4] or the point on the trajectory $p_{f,g}$ which maximizes proximity to the goal is reached.

Now, suppose $\hat{v}_{f,g}$ is a pure rotation. Our experimental implementations have intersected two C-surfaces $\ker f$ and $\ker g$ to yield pure rotational paths sliding along the intersection manifold of $\ker f \bigcap \ker g$ (see chapter 4 for the details). In Proposition (4.4), we demonstrate that these paths may be approximated to an arbitrary resolution by successive applications of the local operators, with only a linear increase in the number of path segments as the resolution grows finer. We have also found it useful to approximate the rotational path along the intersection as follows.

Given two level C-surfaces $\ker f$ and $\ker g$ at configuration X, we wish to choose a direction from X tangent to both. For example, if the configuration space were isomorphic to $\Re^3$, then $\ker f$ and $\ker g$ would both be two dimensional surfaces in 3-space, and this direction would be $\nabla f(X) \times \nabla g(X)$. (Where $\times$ denotes the standard cross product on $\Re^3$). In the tangent space to a six-dimensional *C-Space*, there are typically four such tangent vectors at X which are tangent to $\ker f$ and $\ker g$. We will demonstrate an operator analogous to $\times$ which produces one such tangent vector in a natural way. (It is also possible to solve for all such tangent vectors).

We begin be defining an extended product on the tangent space to $\Re^3 \times SO(3)$ at X. Let $V = (V_x, V_\Theta) \in T_X$ be a tangent vector at X. We may think of V_x and V_Θ as the translational and rotational components of a six-dimensional velocity

[4]Although we also employ the backing off heuristic here.

vector V at X. If $W = (W_x, W_\Theta) \in T_X$ is another tangent vector at X, we define the extended product of V and W by

$$V \hat{\times} W = (V_x \times W_x, V_\Theta \times W_\Theta).$$

The cross products on the right hand side are simply the standard three-dimensional cross product. (See below (2.3) for why this makes sense for the rotational components, $V_\Theta \times W_\Theta$). If $V = \nabla f$ and $W = \nabla g$ then $V \hat{\times} W$ is tangent to both $\ker f$ and $\ker g$ at X. Since $\hat{\times}$ only operates on tangent vectors to $\Re^3 \times SO(3)$ which have the same point of application, we will never have reason to confuse it with $\times$, which can only be applied to three-dimensional tangent vectors.

Let $f, g \in \Lambda'$ be C-functions generating the C-surfaces $\ker f$ and $\ker g$ at X. Observe that the tangent vector $\nabla f(X) \hat{\times} \nabla g(X)$ is tangent to both $\ker f$ and $\ker g$ at X. We can locally approximate a pure rotational trajectory sliding along the intersection of f and g by a path in direction

$$\pi_\Theta(\nabla f(X)) \times \pi_\Theta(\nabla g(X)). \tag{2.3}$$

Note that this is well defined since

$$\pi_\Theta\left(\left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}, \frac{\partial f}{\partial \psi}, \frac{\partial f}{\partial \theta}, \frac{\partial f}{\partial \phi}\right)\right) = \left(\frac{\partial f}{\partial \psi}, \frac{\partial f}{\partial \theta}, \frac{\partial f}{\partial \phi}\right).$$

The differential rotations from X are isomorphic to a three dimensional vector space, and hence the cross product

$$\left(\frac{\partial f}{\partial \psi}(X), \frac{\partial f}{\partial \theta}(X), \frac{\partial f}{\partial \phi}(X)\right) \times \left(\frac{\partial g}{\partial \psi}(X), \frac{\partial g}{\partial \theta}(X), \frac{\partial g}{\partial \phi}(X)\right)$$

is also well defined, and guaranteed to be tangent to $\ker f$ and $\ker g$ at X. The *Rotate* operator can be called in succession on the largest components of (2.3) in order to approximate the sliding trajectory. Of course, it is also possible to re-evaluate the tangents after each step.

2.4.5. The Sliding Expert

The sliding expert attempts to find a path sliding along one level C-surface at X, which makes progress towards the goal. The sliding expert can be thought of as a less constrained version of the intersection expert. The sliding expert tries to choose a C-surface in A' to which the goal-direction is (almost) tangent. As we will see in chapter 4, it is possible to choose a parameterization along a C-surface which maximizes progress. This path along the C-surface can then be realized (at a desired resolution) by successive applications of the local operators. However since there are many paths from X sliding along a C-surface at X, we need to develop a good heuristic strategy.

Our motivation is as follows. There are an uncountable number of paths from X sliding along a C-surface at X. We could maximize a directional derivative at X to choose a locally optimal search direction. This would work once; however, this would not solve the problem of state: it is necessary to partition the set of paths into "neighborhoods," and to mark a neighborhood of paths as explored when a representative from that neighborhood is selected and attempted by a local operator. In principle, a computation involving homotopic equivalence classes is possible (see Donald (1983a) and appendix III). However, this requires a global computation in *C-Space*. In particular, the image of all paths in an equivalence class may cover $\Re^3 \times SO(3)$, even if there are several classes. We wish to find a way to partition the paths from X into neighborhoods, sample a canonical element from the neighborhood, and evaluate it as a local move in the search.

Given a C-surface normal ∇f at X, we wish to choose a direction $\hat{v}$ sliding along the C-surface $\ker f$ which maximizes progress to a subgoal. Let $B = (\hat{x}, \hat{y}, \hat{z}, \hat{\psi}, \hat{\theta}, \hat{\phi})$ be the obvious orthonormal basis for the tangent space to $\Re^3 \times SO(3)$, and $-B = (-\hat{x}, -\hat{y}, -\hat{z}, -\hat{\psi}, -\hat{\theta}, -\hat{\phi})$.

Next, we form a set of vectors orthogonal to $\nabla f(X)$ as follows:

$$D = \{\nabla f(X)\} \otimes \left(B \bigcup -B\right)$$

where $\mathcal{P} \otimes \mathcal{Q} = \{p \hat{\times} q \mid p \in \mathcal{P}, q \in \mathcal{Q}\}$. All of these vectors are orthogonal to $\ker f$ at X. We then choose the direction $\hat{v} \in D$ which maximizes $\Phi_X(\hat{v}, (G - X))$, where the $G - X$ is the goal direction. If Φ_X is the heuristic product on tangent vectors insead of the single-valued Riemannian inner product, then both components of the image of Φ_X should be maximized. In chapter 4 , we will see that it is possible to comply as closely as desired to the C-surface $\ker f$ while traveling in direction $\hat{v}$.

To understand this strategy, consider the following example: Suppose we employ a basis $\mathcal{B}'$ which only spans $\Re^3$. Then the expert will choose the available translation sliding along the level C-surface which maximizes progress towards the goal. Once the direction $\hat{v}$ is chosen, the *Translate* operator is invoked to slide along the level C-surface until a constraint is reached.

There is no need for the basis $\mathcal{B}$ to be orthogonal; this was merely adopted for the sake of intuitive development. The basis provides a sampling of the function space of paths compliant to the C-surface about X.

A Conjecture on Completeness using Extended Spanning Sets

By using the basis $\mathcal{B}$, we obtain a 12-way sampling of the space of directions orthogonal to ∇f at X—in other words, there are 12 vectors in D. Imagine using another set of vectors, $\mathcal{B}^+$, which is larger than $\mathcal{B}$, to construct D. Then D would provide a *finer* sample of the space of directions, since more directions would be sampled. In principle it should be possible for a sample to be complete at a given resolution. We formalize this idea as follows:

A *spanning set* for a space V is a set of vectors which spans V yet which is not necessarily a basis. A spanning set is a basis for V which has been extended by adding other vectors. We conjecture that there exist certain spanning sets which might be employed to construct a complete planning algorithm *without* the Bumble strategy. What constitutes such a complete spanning set? The analogue of resolution for an arbitrary spanning set $\mathcal{B}^+$ would consist in (1) the cardinality of the spanning set and (2) the uniformity of distribution of the vectors

$$\mathcal{B}^+ \bigcup -\mathcal{B}^+$$

about the unit five-dimensional sphere S^5 in the tangent space at X. The greater the number of vectors in the spanning set, and the more uniform their distribution about S^5, the finer the resolution of the planner. The development of such a planning algorithm requires surmounting additional theoretical and technical difficulties.

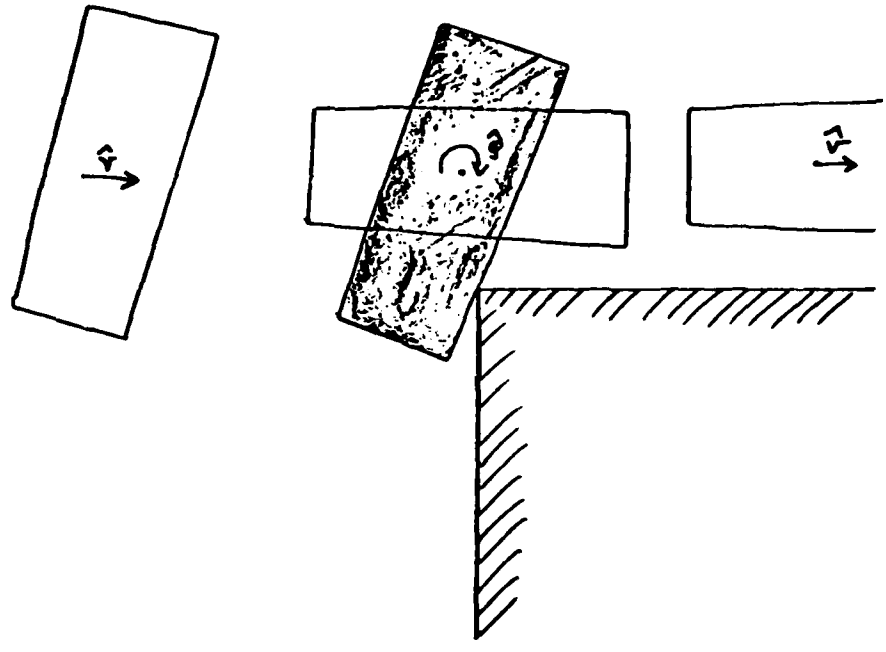


Figure 2.5. An idealized illustration of the hit and rotate strategy. Some expert moves the robot in direction $\hat{v}$ until a C-surface S is hit at configuration X. When the planner tries to plan a move from X, the rotation expert is called to calculate a rotation away from S (in direction $\hat{\phi}$). From the new configuration, direction $\hat{v}$ can be pursued again.

2.4.6. The Rotation Expert

The rotation expert is built on the rotational operator *Rotate*, and is designed to handle some of the special problems of moving through rotation space that are discussed in chapters 3, 4, and 5. The rotation expert might be called to accomplish a simple rotational subgoal, or in conjunction with some more elaborate strategy. In particular, when a translational motion terminates by striking a C-surface, forwarding messages are left for both the sliding expert and the rotation expert. The former has been discussed as the "hit and slide" strategy (figure 2.4); the latter is known as the "hit and rotate" technique (figure 2.5).

The first problem that the rotation expert must deal with is the "wrap around" in rotation space. A subgoal ϕ_0 can be reached in directions $+\hat{\phi}$ and $-\hat{\phi}$, although typically one is "shorter". In conjunction with the planning history, the rotation

expert, on successive applications to the same node, can develop strategies for rocking back and forth on a slice of rotation space.

The *Rotate* operator is more constrained than the *Translate* operator (in that it can only be applied in $\pm\hat{\psi}$, $\pm\hat{\theta}$, and $\pm\hat{\phi}$). Hence the rotation expert must have a method for approximating rotational trajectories (specified in angle space) which are linear combinations of the rotational basis vectors, such as

$$\hat{v} = a\hat{\psi} + b\hat{\theta} + c\hat{\phi} \tag{2.4}$$

for some scalars a , b, and c.

In terms of the completeness of the algorithm, there is no need for a rotate operator in direction (2.4) (provided a path along $\hat{v}$ lies in open sets of free space). In chapter 4, we show that a continuous path may be approximated as closely as desired by a sequence of moves along the rotational axes, and that the number of staggered path segments required grows only linearly as the resolution becomes finer. In practice this use of the restricted rotate operator has proved adequate in our path-finding experiments. However, it is heuristically useful to realize such paths as accurately as desired, since this allows higher level experts to suggest arbitrary rotational trajectories. Given such a trajectory, the rotational directions are ranked by magnitude of change, and the unexplored direction of greatest change is first attempted. On failure, or upon successive applications of the rotation expert to the search node, the other directions in (2.4) will be attempted. This process leads to the approximation of arbitrary pure rotations by a staggered sequence of rotations along the axes. If the extent of each rotation is limited, the approximation can be made arbitrarily fine. To approximate motion in a direction such as (2.4), the planner actually attempts several of the directions simultaneously, which results in a spanning "box" of rotational moves about the idealized trajectory (in the absence of obstacles).

Suppose a, b, and c in the idealized trajectory (2.4) are positive. This yields a set of positive, or "forward" rotational directions, and a set of "backwards" rotational directions which can attain the goal. Which directions are forward and

which are backward depend upon the distance (in the vector parameter space V) of the goal from X, that is, on $\pi_\Theta(G - X)$. For example, if $G_\phi - X_\phi$ is negative and small, then $+\hat{\phi}$ will be a backwards direction, and $-\hat{\phi}$ will be a forward direction.

The rotation expert develops and ranks these sets of forward and backward rotational directions. By examining the planning history and the local geometry of C-surfaces at X, these sets of directions are in turn pruned. In particular, local C-surfaces that would block a particular rotational motion are detected. For a direction $\hat{v}$, this is done by examining the magnitude of the directional derivative in $\hat{v}$. The importance of such an impediment is then heuristically ranked by the closeness of the C-surface at X. Special consideration is given to C-surfaces which have a history of proving troublesome. For example, when an expert runs into a C-surface, the reason for stopping is left as part of the move explanation. If the rotation expert is invoked as part of a "hit and rotate" strategy, then we must ensure that the planner tries to rotate away from the C-surface(s) which blocked progress. The rotational directions which point away from C-surfaces may be found by examining ∇f. The process of determining the rotational constraints from the local geometry of C-surfaces is closely related to our earlier discussion of detecting rotationally orthogonal C-surfaces.

Thus the requested rotational trajectory and rotational goal provide a set of desired rotational motions. The planning history supplies a set of rotational constraints, and from the local C-surface geometry can be inferred a set of preferred and prohibited motions. The constraints, preferences, and prohibitions are intersected with the forward and backward desires. This yields a set of rotational directions which will be attempted using the *Rotate* operator. Depending on the kind of invocation, the rotation expert may apply the *Rotate* operator up to some fixed number of times--this is particularly useful when it must attempt to approximate an idealized rotational trajectory which is a linear combination of the basic rotational directions.

Canny (1984) has recently extended the *Rotate* operator for directions such as eq. (2.4), corresponding to uniform rotation.

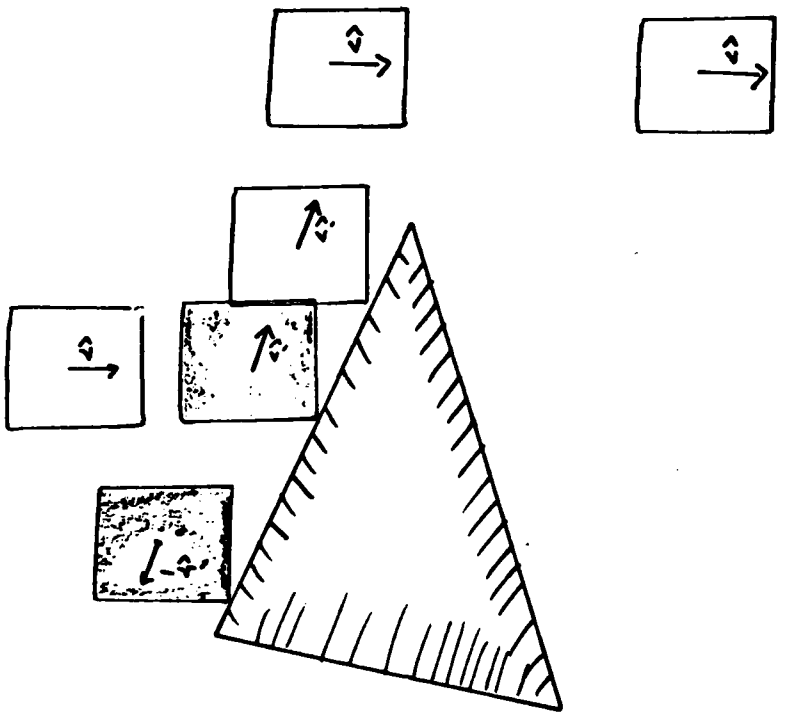


Figure 2.6. An idealized illustration of the around expert. When progress for the moving object in the goal direction $\hat{v}$ is blocked, the expert attempts to find a C-surface which is roughly orthogonal to $\hat{v}$. A sliding motion (either $\hat{v}$ or $-\hat{v}$) is then planned along this level C-surface (around the obstacle). The resulting search node is then expanded.

2.4.7. The Around Expert

The around expert attempts to circumnavigate obstacles by sliding around their boundary. An idealized illustration of the around expert is shown in figure 2.6. The around expert is similar to the sliding expert, except that instead of attempting to find a C-surface which contains a path *towards* the goal, the around expert searches for a C-surface which is (roughly) *locally orthogonal* to the goal direction. Next a path is planned sliding along this surface in a direction $\hat{v}'$ orthogonal to the goal direction; the path is attempted using a local operator. Typically, this motion will result in a search node s' which is farther from the goal than the parent node, s. Ordinarily, s' would not be explored soon, since other search nodes would appear more promising to the planner's best-first strategy. In order to give the around strategy a chance, the around expert explicitly places s' at the front of the search

queue and calls the planner recursively.

The around expert can also invoke the intersection expert. Recall that the intersection expert normally tries to construct tangent intersection manifolds which contain paths towards the goal. However, when called from the around strategy, it can construct intersection manifolds locally orthogonal to the goal direction. To construct the intersection set of locally orthogonal level C-manifolds, we perform a pairwise intersection of C-manifolds locally orthogonal to the goal direction at X.

2.4.8. The Suggestor

The suggestor is a strategy for proposing good subgoals in configuration space. As we saw in Donald (1983a), one of the problems with local operators even if they are complete (that is, their closure covers configuration space), is that without good subgoals, they may take a long time to converge. The suggestor is a heuristic strategy for setting subgoals in *C-Space*.

First, a very coarse lattice is thrown over *C-Space*. This lattice is then searched for a *sequence* Q of free configurations (*not* a path) stepping through the lattice to the goal. If no such sequence can be found, then configurations on a promising partial sequence are employed. These configurations may then be set as subgoals, and the planner can be called recursively. The configurations Q represent intermediate planning islands of safe configurations. If paths can be found between these configurations, then the find-path problem is solved. Otherwise, expanding from any partial paths found can also prove useful, in that the planning islands effectively distribute the application of local experts and operators over more of configuration space.

The suggestor complicates the connectivity of the explored neighborhoods graph. The ability to explore arbitrary subgoals and suggested paths requires more complicated bookkeeping for neighborhood exploration: we must employ the *connect strategy*, in order to know when partial paths link up. If partial paths not rooted at the start neighborhood are permitted, then the graph of explored neighborhoods will not necessarily be connected, and the mark strategy will fail (the mark strategy constructs a directed, spanning tree for a connected, rooted graph

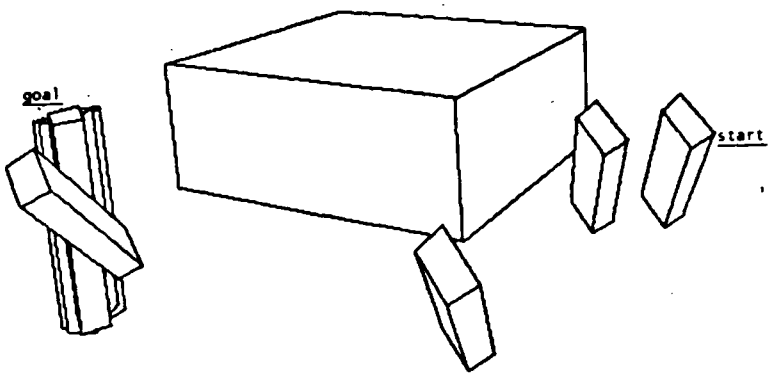


Figure 2.7. A path which was found using local experts. This find-path problem is very easy (it is used as an example in chapter 1).

of explored neighborhoods). Happily the connect strategy will succeed, since it is defined on an arbitrary graph. An algorithm for the connect strategy is discussed in section 2.1.2.

Listing I: The log of expert explanations for the path in figure 2.7.

```
(find-path *s1 *g1)
 Verifying the start and goal points...
 start : (0 0 0 1 1 11), goal : (-6 10 0 0 0 0).
Starting search, boss...
Exploring (0 0 0 1 1 11)...
Local Expert: I translated straight towards goal, reaching ((-1 1 0 1 1 11))
Exploring (-1 1 0 1 1 11)...
Local Expert: I Slid along a level C-Manifold, reaching ((-6 1 0 1 1 11))
Exploring (-6 1 0 1 1 11)...
Local Expert: I translated straight towards goal, reaching ((-6 10 0 1 1 11))
Exploring (-6 10 0 1 1 11)...
Rotation-Expert: Found 0 guiding constraints on rotational motion.
Rotation-Expert: Intersected Rotational Constraints with desired
                 rotations yielding possible motions in
                 ((MINUS PHI) (MINUS PSI) THETA).
Rotation-Expert: I am trying to rotate in (PLUS THETA) ...
Local Expert:        I rotated  to reach ((-6 10 0 1 1 0))


Exploring (-6 10 0 1 1 0)...
Rotation-Expert: Found 0 guiding constraints on rotational motion.
Rotation-Expert: Intersected Rotational Constraints with desired
                 rotations yielding possible motions in
                 ((MINUS PHI) (MINUS PSI)).
Rotation-Expert: I am trying to rotate in (MINUS PHI) ...
Local Expert:        I rotated  to reach ((-6 10 0 0 1 0))
Exploring (-6 10 0 0 1 0)...
Rotation-Expert: Found 0 guiding constraints on rotational motion.
Rotation-Expert: Intersected Rotational Constraints with desired
                 rotations yielding possible motions in
                 ((MINUS PSI)).
Rotation-Expert: I am trying to rotate in (MINUS PSI) ...
Local Expert:        I rotated  to reach ((-6 10 0 0 0 0))
Exploring (-6 10 0 0 0 0)...
 [success!] Saving and Drawing final path...
Back to Lisp Top Level in Lisp Listener 2
```

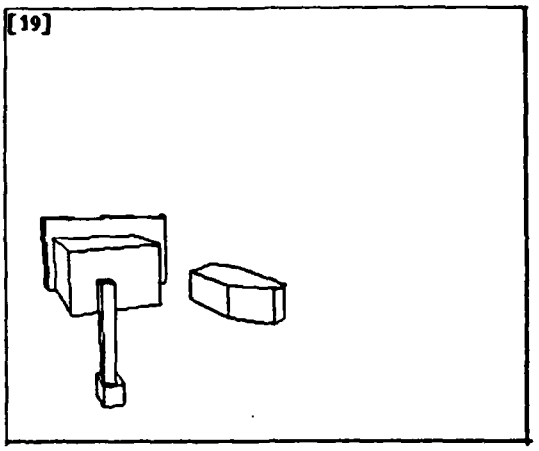


Figure 2.12. View II: (frame 19), The final configuration.

2.5. Examples of the Local Experts in Use

In figure 2.7, we show a very simple example of a path found using local experts. Listing I shows a log of the expert explanations for each move.

The "Thor's Hammer" example in chapter 1 was produced by disabling all experts, and employing only the Bumble strategy. (Please refer to this figure). In the accompanying figures (2.8–13), we show a path found by a strategy comprising all the experts described above. The solution path is very different, and tends to slide around obstacles instead of finding convoluted paths between them.

Figures 2.14–21 show the solution for a find-path problem in a cartesian workspace. A *cartesian workspace* is a bounding box beyond which the reference point may not translate. However, the bounding box imposes no restrictions on rotations. The Movers' problem in a cartesian workspace is similar to the motion-planning problem for cartesian manipulators, and the L-shaped object may be thought of as the (wrist and) payload. First, we show the reference point on the L-shaped object. Next two views are presented of the path found within the workspace, around a large, diagonally-placed obstacle. View (II) is a view from the side; view (I) is a view from the top. Only the back faces of the rectangloid workspace are shown. Since the rotation from frames 13 to 14 is very large ($> \pi$

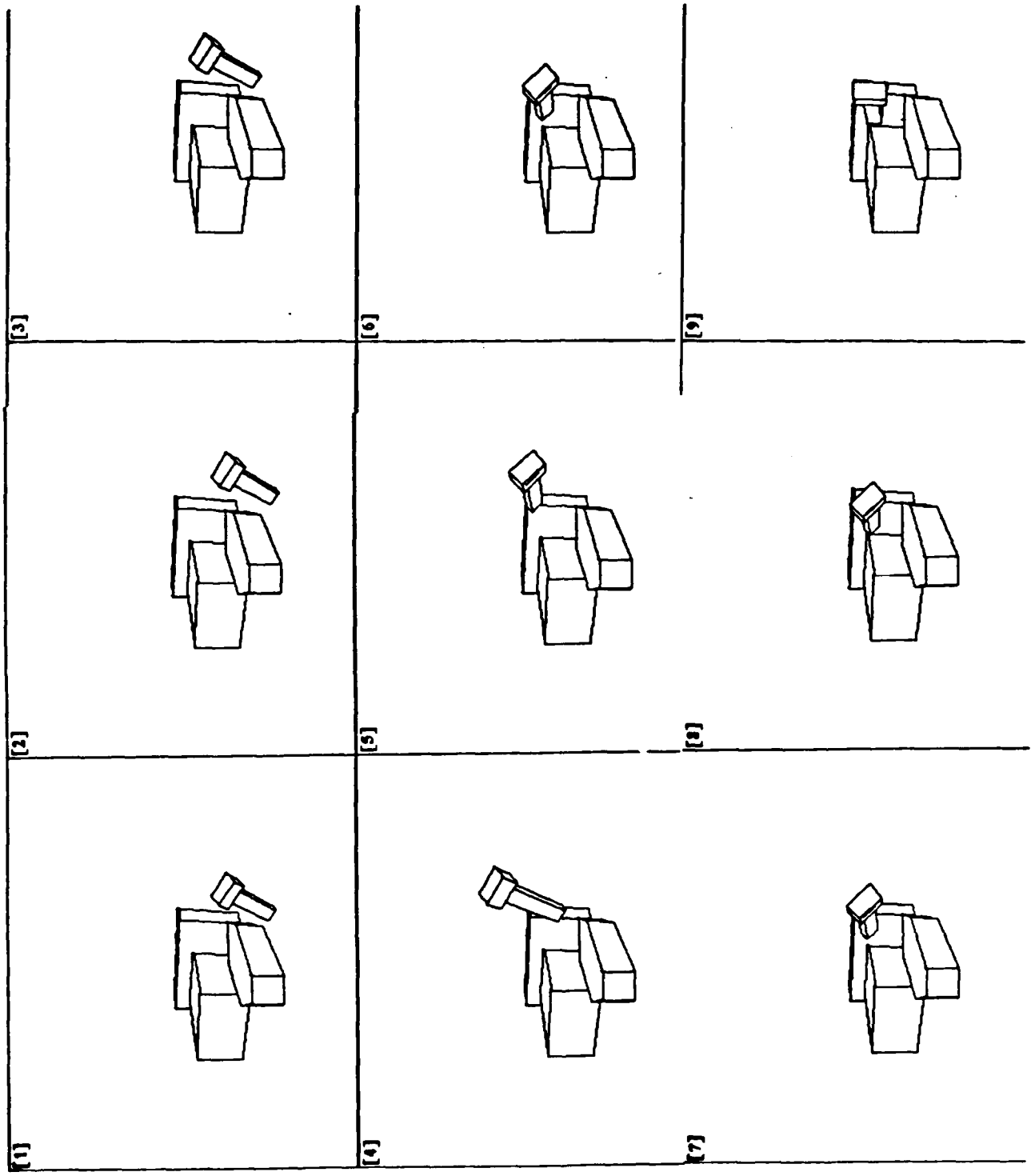


Figure 2.8. View I: (*frames 1 9*). These 18 frames show a solution path for the "Thor's Hammer" Mover's problem. Local experts (as described in this chapter) are employed to slide the moving object along level C-Manifolds. Three views are shown. The final configuration is only visible in view II (figure 2.12).

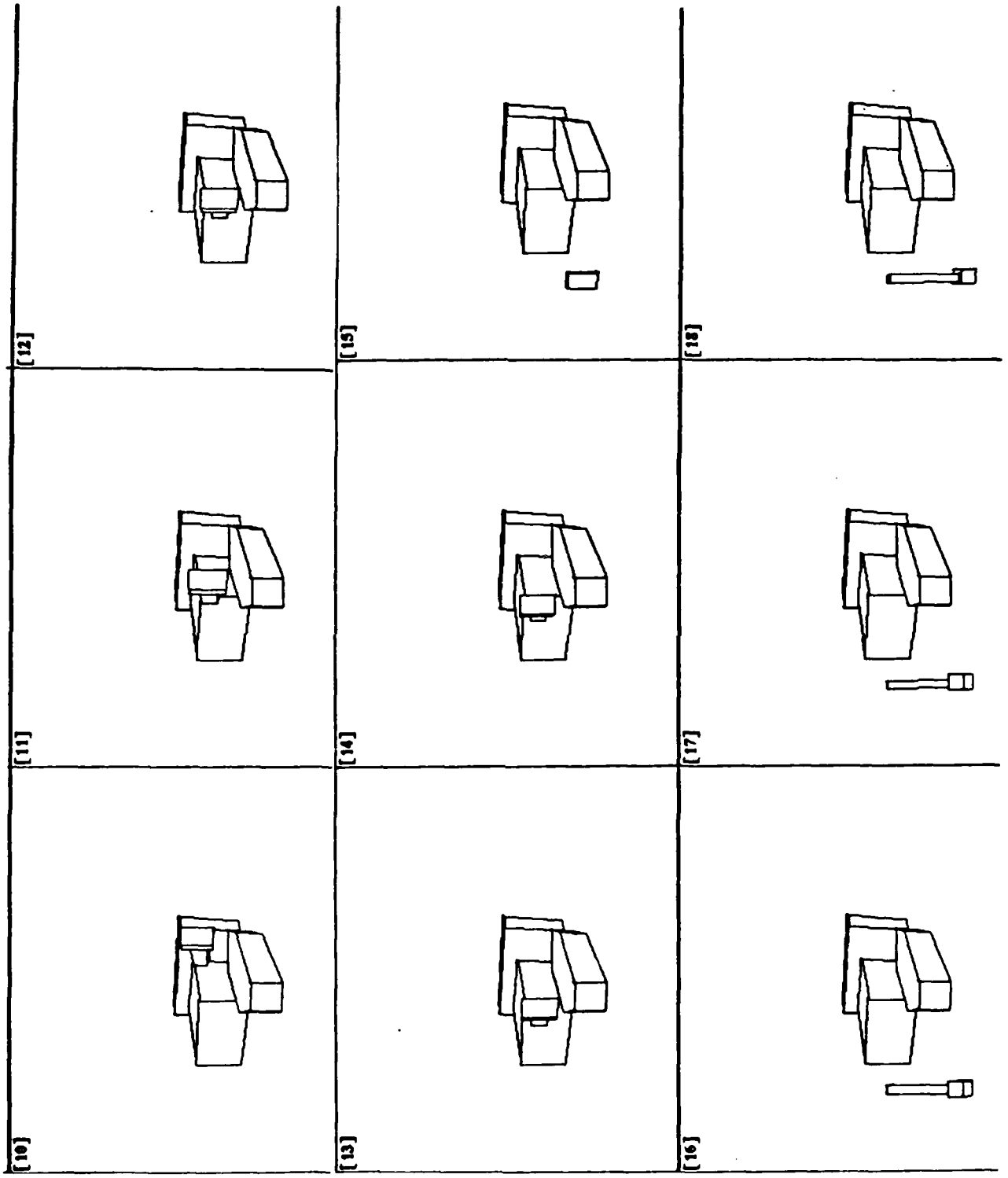


Figure 2.9. View 1: (frames 10-18).

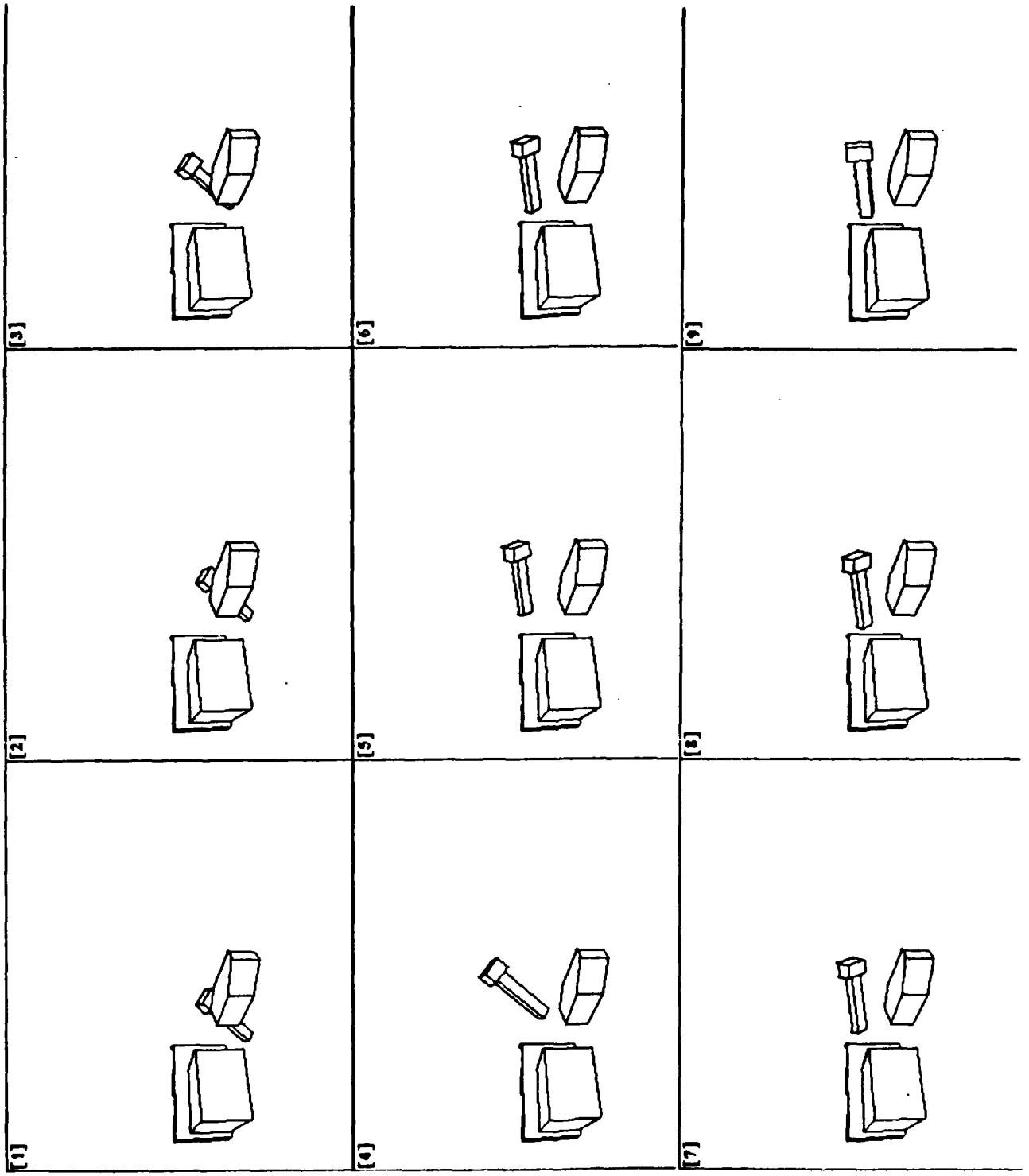


Figure 2.10. View II (frames 1 9) of the Thor's hammer Example using Local Experts.

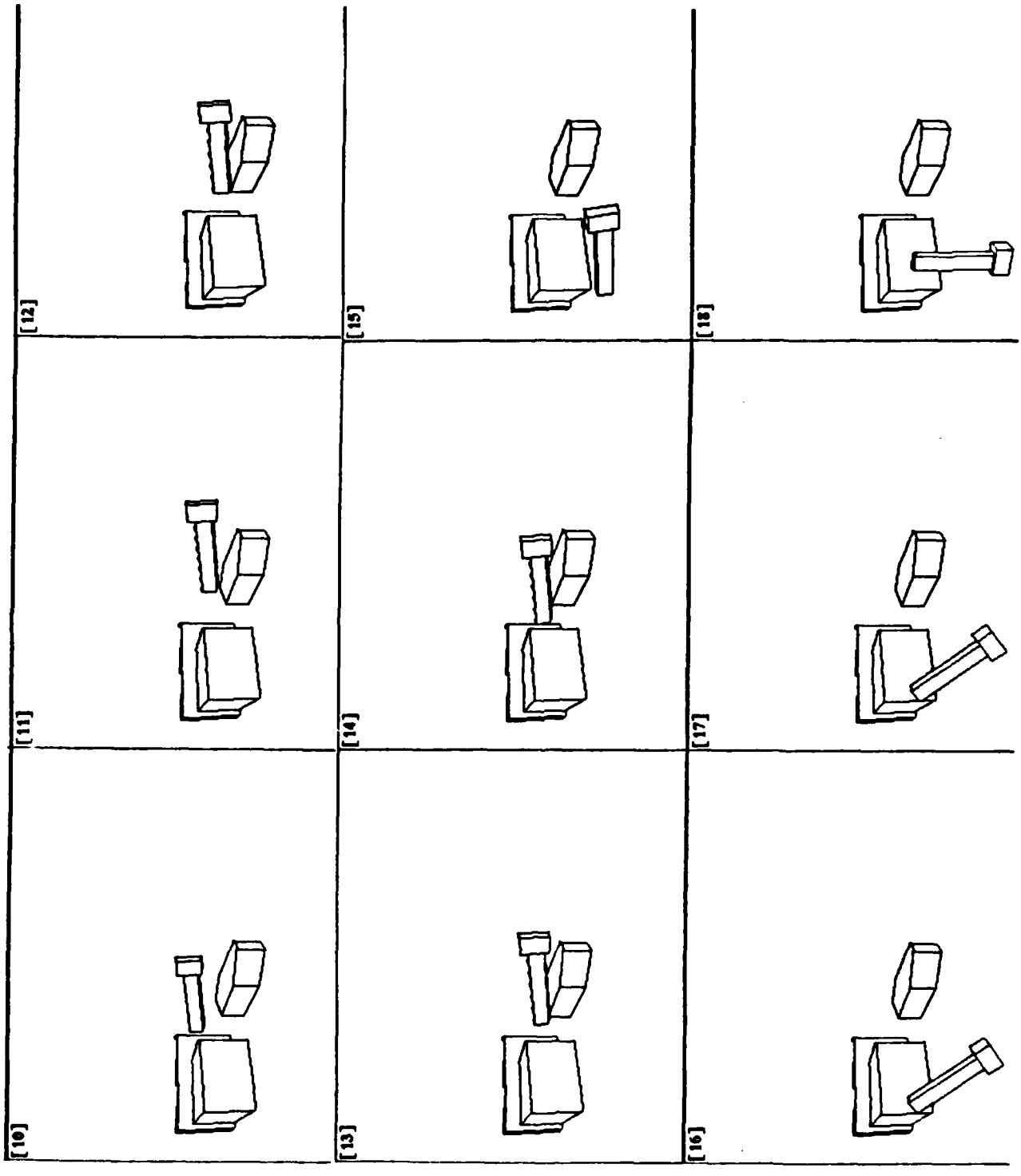


Figure 2.11. View II: (frames 10 18).

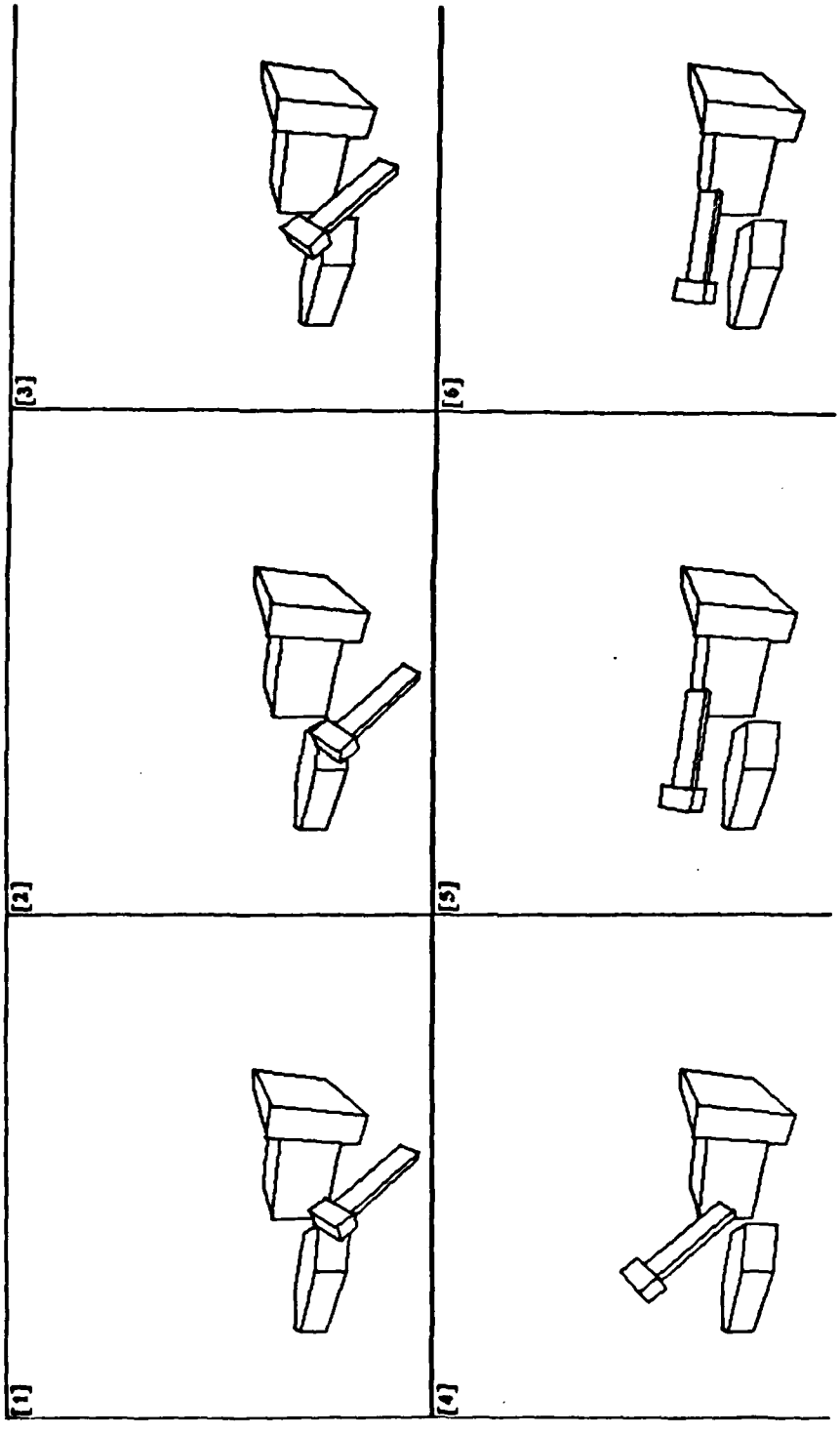


Figure 2.13. View III: (A detail of frames 1 through 6).

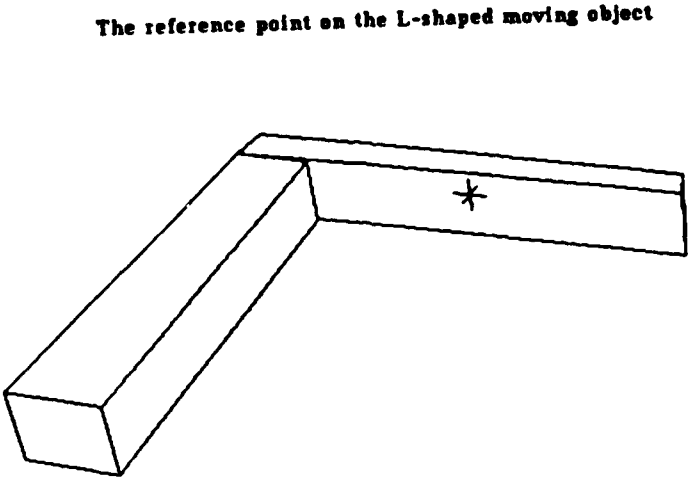


Figure 2.14. The reference point on the L-shaped object.

in the $-\hat{\psi}$ direction), a detail of the rotation is also shown.

2.6. Path Planning versus Discrete Intersection Detection

Imagine a brute-force planner which discretizes configuration space, places the robot at every point in the discretization, and tests for intersection. This would yield a discrete set of configurations where the robot could be placed. Alternatively, the tests could be structured in a search. As stated so far, this is not collision-free path planning. Path planning ensures that a path exists between each configuration on the path. It has been argued that if the intersection-detection is done at a fine enough resolution then a path will have been effectively found. At a given resolution, it is possible to bound the size of the intersection between the robot and any obstacle which can occur between intersection checks. This bound grows smaller as the sampling of the space grows finer. By growing the real-space obstacles by this bound, it is possible to ensure that no collisions occur between discrete

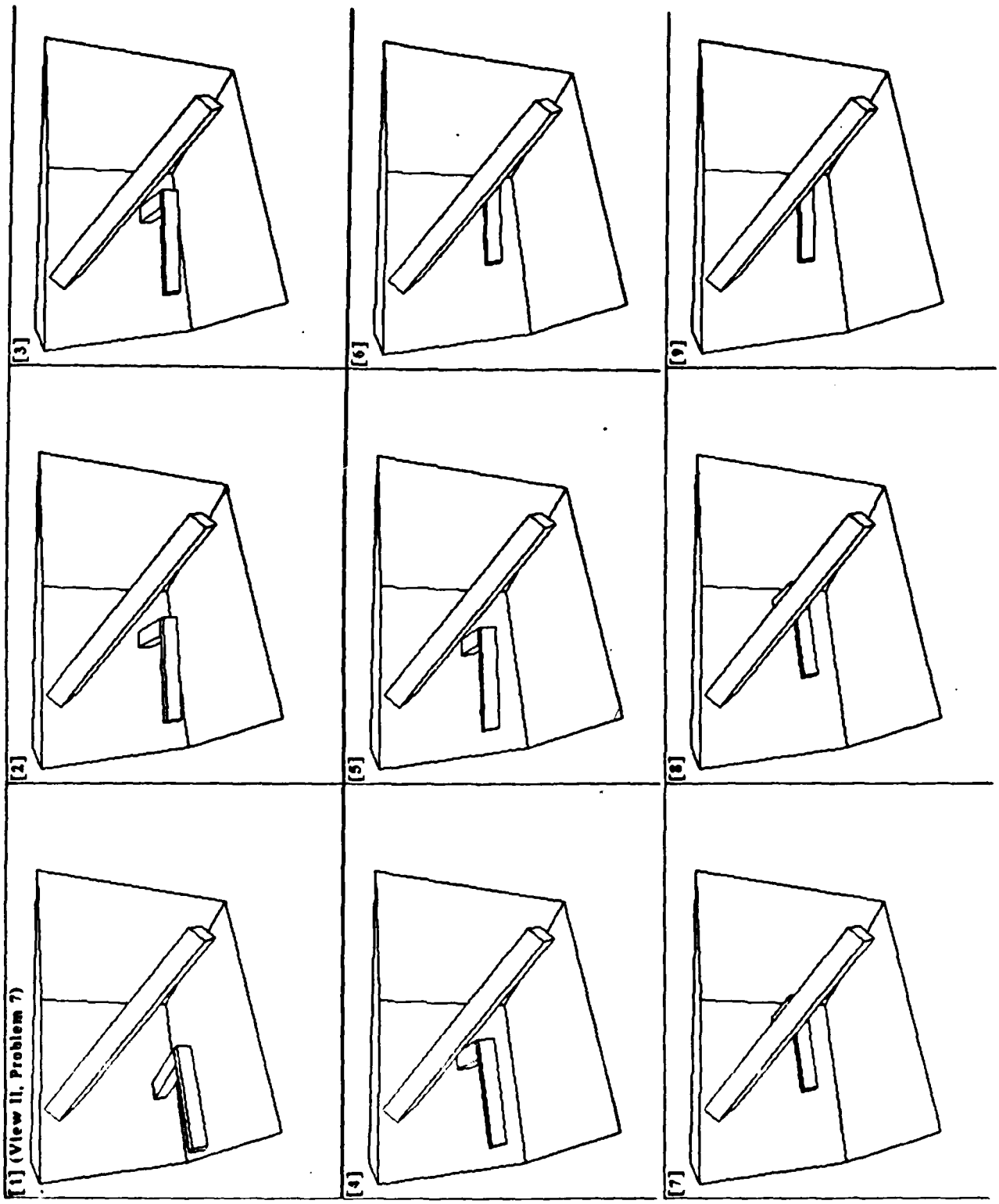


Figure 2.15. Solution Path, View (II), frames (1-9)

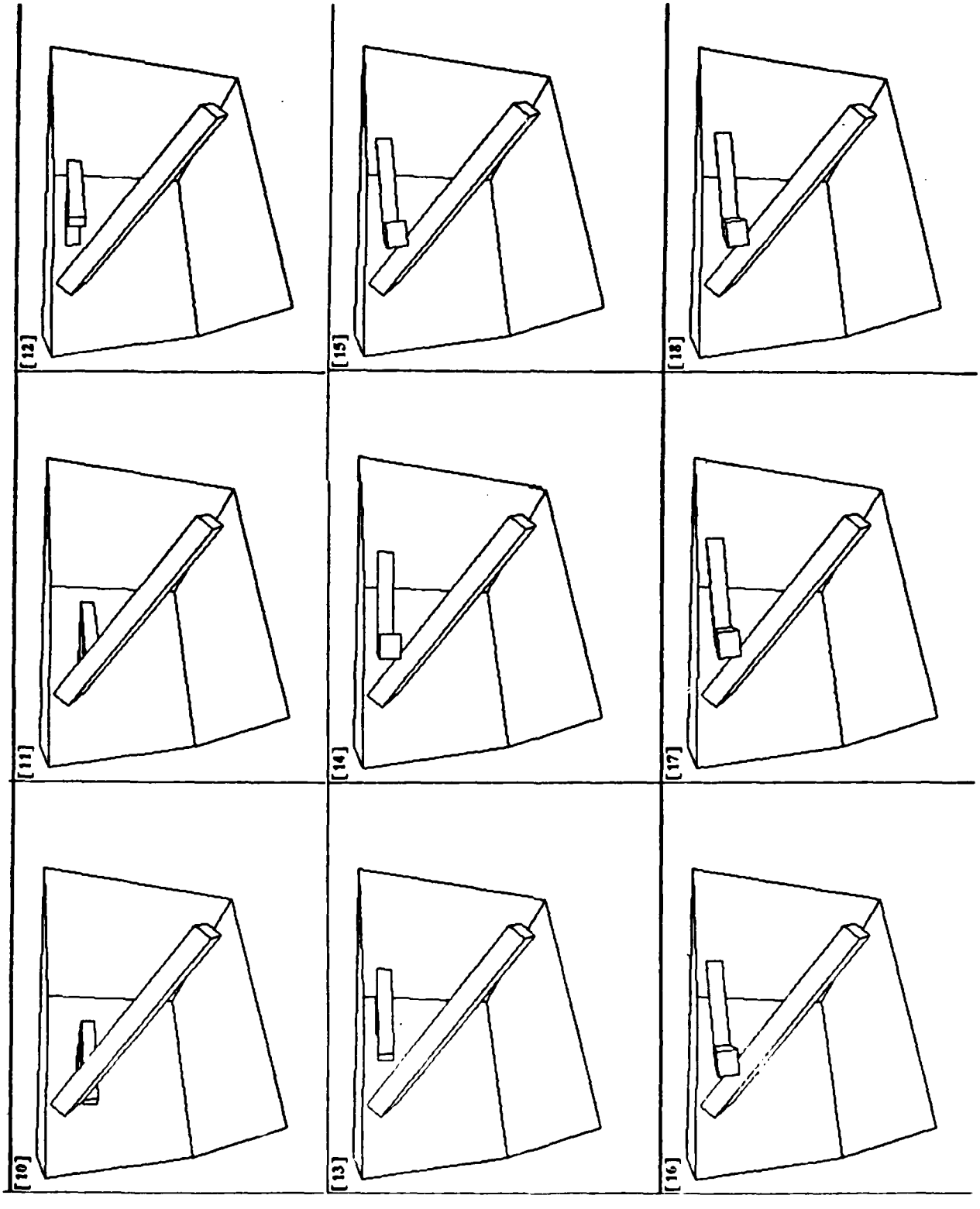


Figure 2.16. Solution Path, View (II), frames (10-18)

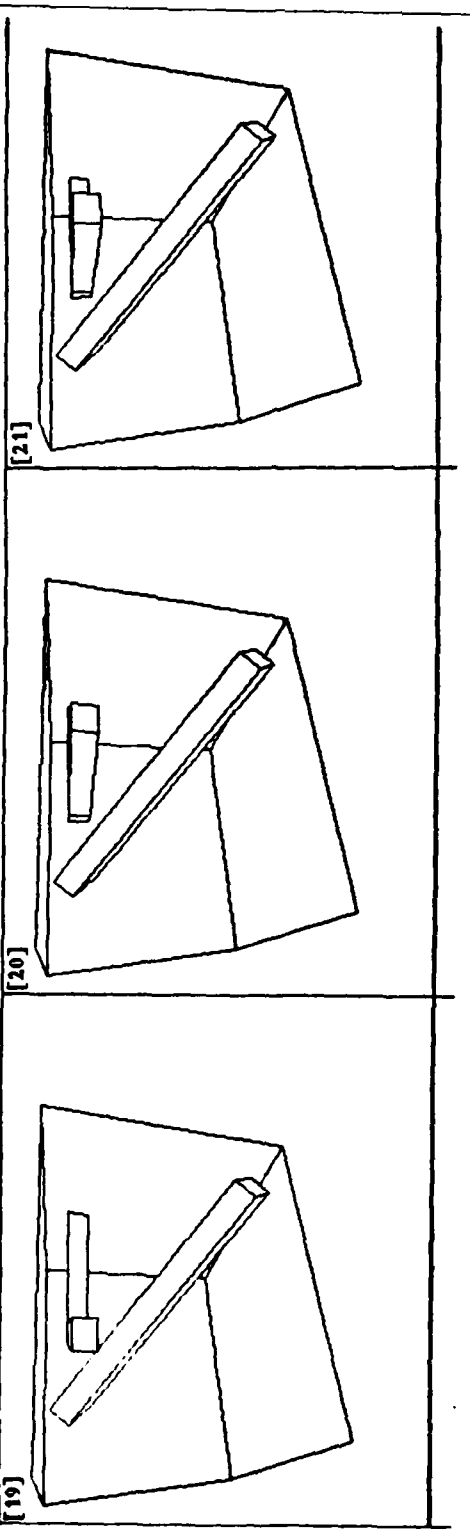


Figure 2.17. Solution Path, View (II), frames (19-21)

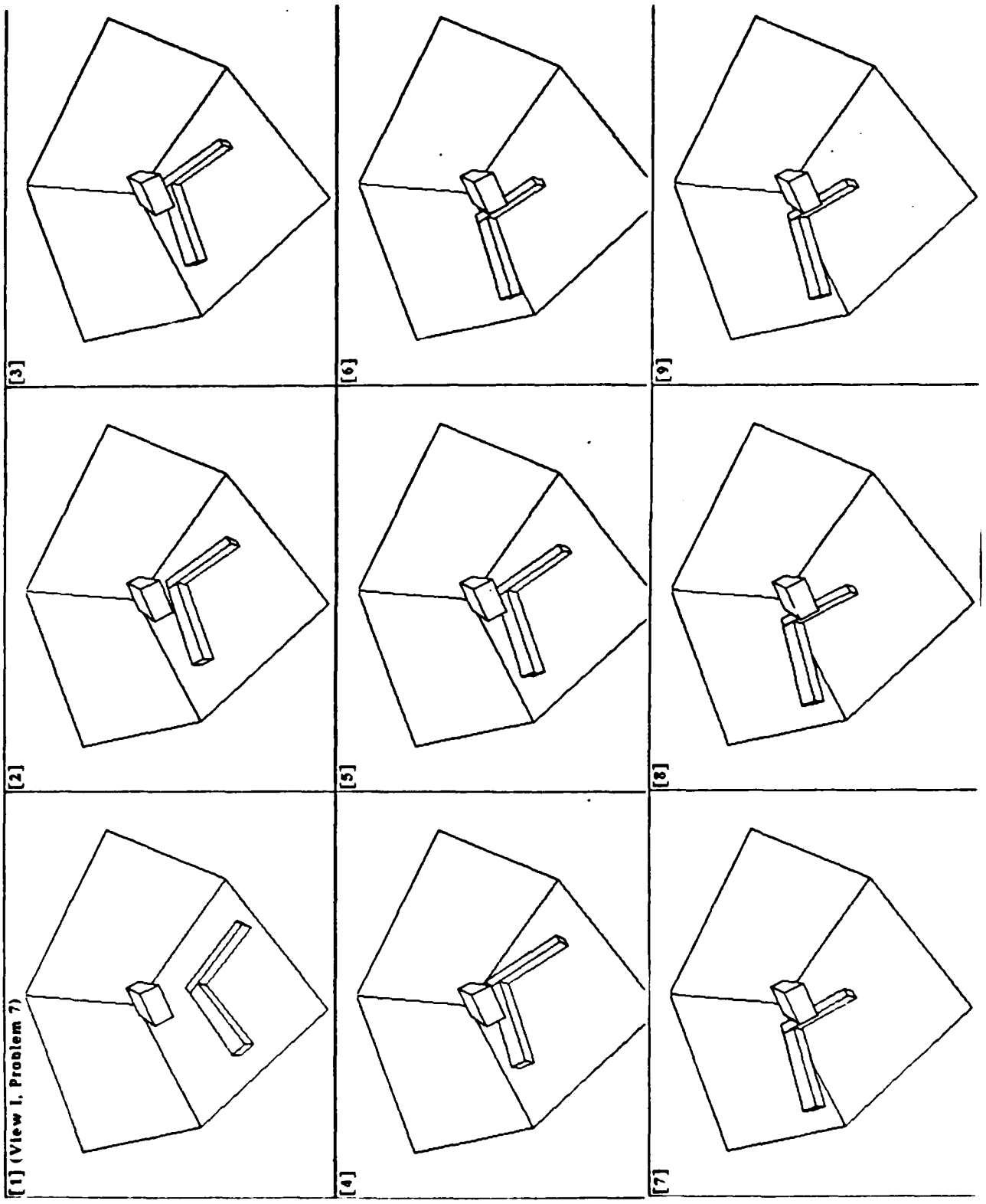


Figure 2.18. Solution Path, View (I), frames (1–9)

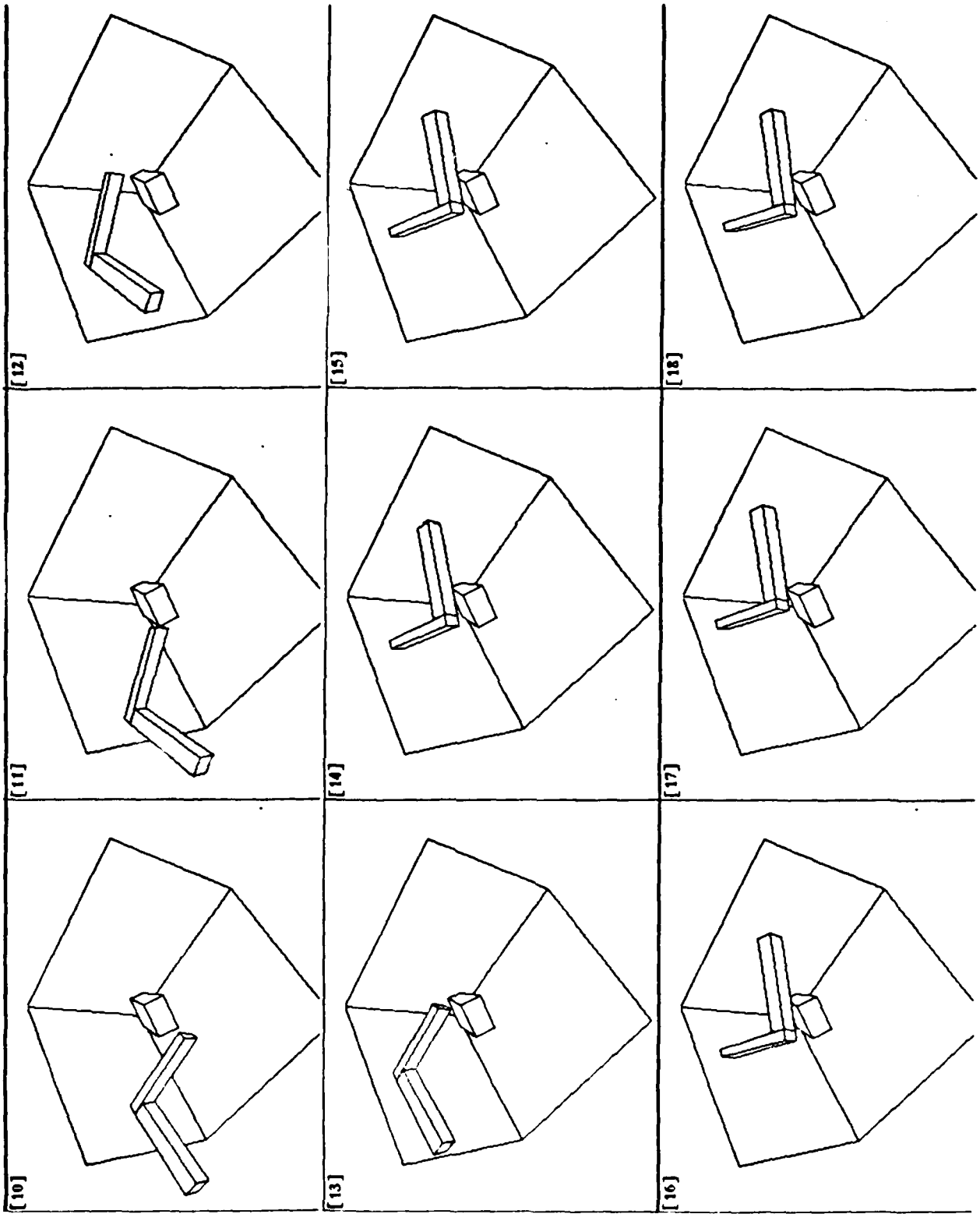


Figure 2.19. Solution Path, View (1), frames (10-18)

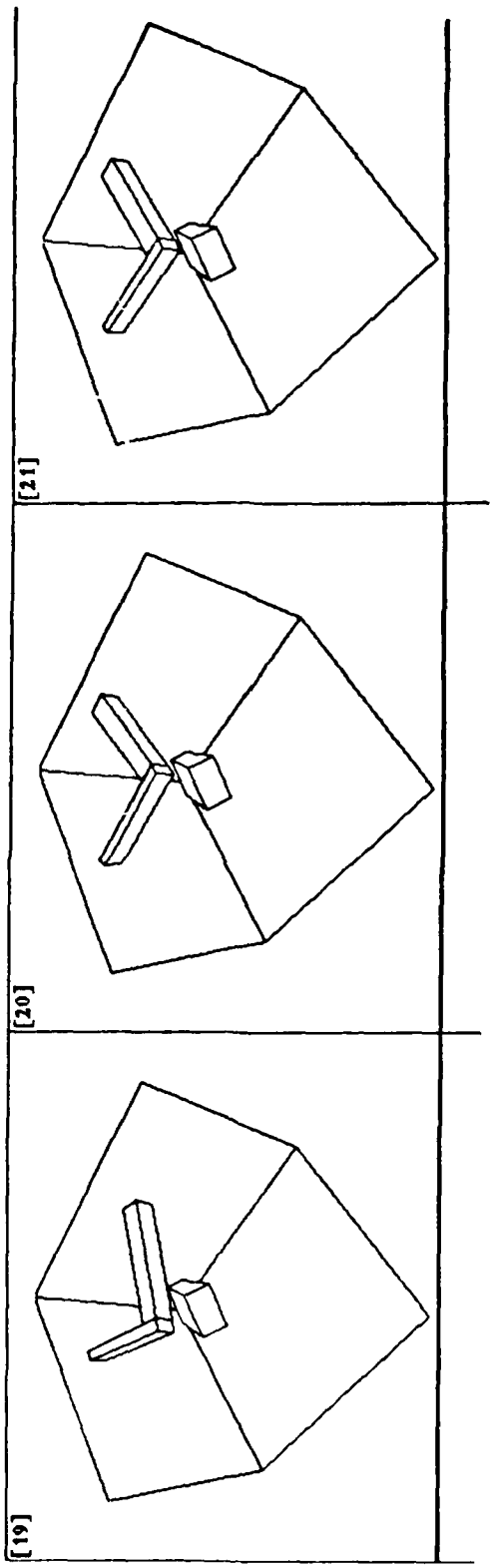


Figure 2.20. Solution Path, View (1), frames (19 21)

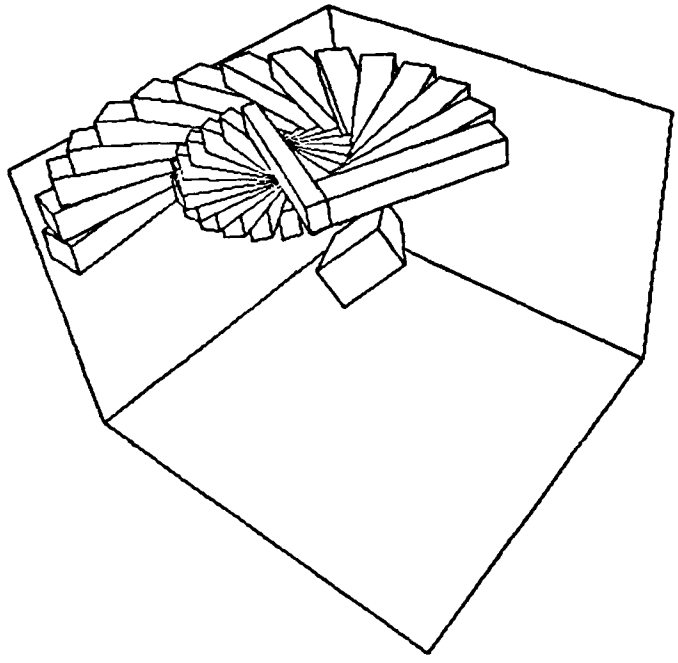


Figure 2.21. Detail of the rotation from frames 13–14.

intersection checks (Gouzenes (1983)). Of course, if the resolution is insufficiently fine, then the obstacles may be grown so much that no path can be found.

For gross motion planning in an uncluttered environment, this approximate method may perform reasonably well. In complicated environments, however, the resolution will have to be fine in order to ensure that paths are collision free without growing the obstacles so much that no path can be found. We will compare the asymptotic complexity of the discrete intersection method with the *Rotate* operator. (The *Rotate* operator is the most complex local operator). The fundamental observation is that the complexity of the discrete intersection method varies linearly with the sampling resolution, whereas the complexity of the *Rotate* operator is independent of (any) resolution. This is because our discretization is quite different: a lattice is thrown on the space in order to record the *state* of the planner and the *connectivity* of the explored neighborhoods.

Consider the following. Suppose X and Y are configurations on a rotational trajectory in direction $\hat{\phi}$. Suppose further that the robot is composed of m convex polyhedra containing k generators each, and that there are n convex obstacles containing j generators each. The number of faces on a robot polyhedron or an obstacle polyhedron is $O(k)$ (respectively, $O(j)$). To perform one intersection check (at a single configuration) for one robot polyhedron and one obstacle polyhedron requires time $O(\log^2(j+k))$ (Dobkin and Kirkpatrick (1980)). This theoretical intersection algorithm has not yet been implemented, but we consider it since it is the fastest known. To perform a check (at one configuration) for the entire robot against the entire obstacle environment requires time $O(mn\log^2(j+k))$. Now suppose that the path segment $[X,Y]$ must be sampled i times for the quantizing intersection checker. This requires time

$$O(imn\log^2(j+k)).$$

In chapter 5, we show that our planner's *Rotate* operator could determine whether there exists a path from X to Y in time $O(N\log N)$ (where N is the number of C-surfaces). In chapter 3, we show that $N = O(mnjk)$. Hence the complexity for *Rotate* is

$$O(mnjk\log(mnjk)) = O(mnjk(\log m + \log n + \log j + \log k)).$$

Rotate ensures that there exists a path from X to Y without growing the real-space obstacles, and does not involve a resolution factor i. Holding k and j fixed, the relative asymptotic performance of the quantizing intersection detector and the *Rotate* operator will depend on whether or not

$$i > \log(mn).$$

The constants i, j, and k will depend on particular workspaces and find-path problems. However, we believe that in order to be reasonably sure of the safeness

of a path between configurations without growing the real-space obstacles too much, i may have to be quite large. This is especially true in reasonable sized environments. So as mn increases, the workspace becomes more crowded and/or the robot becomes more complicated, and the sampling rate will have to be increased. We think it unlikely that the sampling rate will grow only logarithmically with the workspace complexity. Moreover, the theoretical $O(\log^2(j+k))$ intersection time for the Dobkin and Kirkpatrick (1980) algorithm assumes that the solid models of the m robot polyhedra are precomputed. (If the solid models must be computed for each configuration, then this will take $O(km)$ additional time per sample point). In addition, for a polyhedron with k faces, $O(k \log k)$ preprocessing time is required by the algorithm (for each intersection check), which would yield an even higher complexity for the discrete path planning algorithm. At this stage, since the algorithm is unimplemented, it is unclear whether some sort of lazy evaluation, parametric representation, or efficient precomputation could be employed to reduce the complexity of iterative application of this intersection test. Most implemented intersection detectors that are reasonably robust have time complexity $O((j+k)\log(j+k))$ or $O((j+k)^2)$. However, it is possible to employ minimum distance checks, or $O(j+k)$ intersection checks in some cases.

Summary

In a practical planning system, there are, of course, other considerations. For example, our employment of the *Rotate* operator requires time to update the lattice. The main point is as follows: on a lattice of spacing d, to verify the safeness of a path of length di, the discrete intersection method requires at least time $O(imn)$, whereas the *Rotate* operator requires time $O(mn \log(mn))$. The discrete method actually does not ensure safeness, but merely that the intersection "size" is no greater than some function of d.

Competence versus Performance

We have shown that the relative performance of the two algorithms will largely depend on the constants in the problem. For gross motion in uncluttered workspaces the discrete intersection algorithm will probably perform better. In complicated, crowded environments, or in problems requiring motions close to the obstacles, the

required sampling rate will probably be prohibitive. In addition to the question of performance, we should also mention the issue of competence. (In linguistics, *competence* refers to the knowledge base, and *performance* refers to how well it is used). The representations we develop in subsequent chapters are applicable not only to the find-path problem with six degrees of freedom, but also to the class of geometric planning problems described in chapter 1 (for example, fine motion, and planning with uncertainty). It is clear from previous work that these problems are within the competence of the representation we develop for $\Re^3 \times SO(3)$ (Mason (1981), Lozano-Pérez, Mason, and Taylor (1983), Erdmann (1984)). At this point we have no indication that these problems are within the competence of the discrete intersection method. (Find-space, however, can be accomplished using discrete intersections).

3
Questions of Representation: C-functions and Applicability Constraints in a Six Dimensional Configuration Space

In this chapter, we first present a formal framework in which several open questions about configuration space constraints may be resolved. This framework has been discussed informally in the first two chapters. We then proceed to construct and prove a set of theorems about the domains and domain topology of C-functions for the classical Movers' problem with six degrees of freedom.

These theorems allow us to define the applicability constraints on C-functions for the Movers' problem in $\Re^3 \times SO(3)$. Every C-function characterizes a constraint on motion only within a certain region of rotation space. Determining what constraints are applicable at a given orientation (or range of orientations) is of fundamental importance to the mathematical framework for the spatial planning problem: *in order to plan using constraints, we must know where (at what orientations) these constraints are applicable.* Recall that each C-function is *generated* by a pair of boundary cells (a, b), where a lies on the boundary of a moving polyhedron and b on the boundary of an obstacle polyhedron. Put simply, the applicability constraints determine what boundary cells a and b can interact at a given orientation.

3.1. Definitions and Conventions

Let A denote any rigid, convex set. $A(\Theta)$ denotes A rotated to orientation Θ. Formally, if Θ is an orientation, and $\mathcal{R}(\Theta)$ is the corresponding rotation operator,

then $A(\Theta)$ denotes $\mathcal{R}(\Theta)$ applied to A. As a kind of shorthand, we refer to $A(\Theta)$ as "A at orientation Θ," or "A rotated to orientation Θ." For example, if F is a face, then $F(\Theta)$ denotes F at orientation Θ. F's normal, N, rotates with F, and is denoted $N(\Theta)$. We assume face normals are outward-directed from the polyhedra they bound. We will in general use A to denote a convex moving polyhedron, and B for a convex obstacle polyhedron. If e_a is an edge of A and $\mathrm{mid}(e_a)$ denotes its midpoint, then $\mathrm{mid}(e_a(\Theta))$ denotes its midpoint at orientation Θ. At this point it is not convenient to commit ourselves to any particular representation for 3-dimensional rotations. However, the reader may without essential loss of generality interpret $v(\Theta)$ (for $v \in \Re^3$) as the rotation matrix $\mathcal{R}(\Theta)$ applied to the vector v, where $\mathcal{R}(\Theta)$ might be parameterized by Euler Angles. Since $\mathcal{R}(\Theta)$ is an orthonormal matrix, $[\mathcal{R}(\Theta)]^{-1} = [\mathcal{R}(\Theta)]^T$ can be employed to rotate a plane which is represented as a 4-dimensional vector. This operation yields the rotated normal $N(\Theta)$ of course (see Paul (1981)). However, note that the results of this chapter are independent of any particular representation of rotations, and that $\mathcal{R}(\Theta)$ is properly a generic rotation operator. $u \cdot v$ denotes the standard inner product on $\Re^3$ of u and v. If u and v are complicated expressions, however, we will use the notation $\langle u, v \rangle$.

The six dimensional configuration space $\Re^3 \times SO(3)$ is formally defined in chapter 2. X will denote a configuration in this space. We will identify Θ with $\mathcal{R}(\Theta)$ and write $\Theta \in SO(3)$. Writing $X = (x, \Theta)$ makes explicit the translational component of the configuration (x) and the rotational component (Θ or $\mathcal{R}(\Theta)$).

∂ denotes the *boundary* operator. For example, if F is a face on a polyhedron B, then ∂F denotes the ring of edges which bound F. ∂B denotes the faces of B, ∂e for an edge e denotes e's vertices, and so forth. The *coboundary* operator is the dual of the boundary operator and is denoted δ. The coboundary of a vertex is the set of edges incident there; the coboundary of an edge are the faces which the edge bounds; and the coboundary of a face is the zero, one, or two solids it bounds. In chapter 5, we provide a formal definition of boundary and coboundary using the chain groups; alternatively, see Hocking and Young (1961) or Giblin (1977).

We denote the faces, edges, and vertices of a polyhedron B by $faces(B)$, $edges(B)$, and $vert(B)$, respectively.

If S is a set then $i(S)$ denotes its interior, and κS its closure. $\kappa S = i(S) \bigcup \partial S$.

We denote the classical Movers' problem with six degrees of freedom by $6DOF$.

3.2. Representing Constraints in Configuration Space

Lozano-Pérez (1983) showed that the *C-Space* obstacles can be represented as an intersection of a finite number of half-hyperspaces,[1] where each half-hyperspace is represented via a constraint function of the form

$$f_i : \Re^3 \times SO(3) \rightarrow \Re$$

where the sign of $f_i(X)$ determines whether X is inside, on, or outside the *C-Space* obstacles. However, when rotations are allowed, each constraint function is valid, or applicable only within a certain region $\mathcal{A}_i$ of the rotation space:

$$f_i : \Re^3 \times \mathcal{A}_i \rightarrow \Re \qquad (\mathcal{A}_i \subset SO(3)).$$

We call such a function f_i a C-function. We consider the robot and obstacles to be modeled by the (possibly overlapping) union of convex polyhedra, and define a *boundary cell* to be a face, edge, or vertex of such a polyhedron. C-functions model constraints on motion generated by pairs of cells (g_a, g_b) where g_a and g_b are boundary cells on the robot and on an obstacle, respectively. Lozano-Pérez (1983) identified three types of interactions: (face,vertex), (vertex,face), and (edge,edge), which to preserve tradition we shall term type (a), (b), and (c) constraints. However, these interactions can only occur in certain orientations; for example, it is easily seen that although two cuboids generate 144 type (c) constraints, at any fixed orientation only certain edges can interact and hence only certain type (c) constraints are applicable. The region of rotation space where a C-function f_i is applicable is it's applicability region, $\mathcal{A}_i$. The domain of f_i, then, is $\Re^3 \times \mathcal{A}_i$.

For the two-dimensional Movers' problem, the rotation space is the 1-sphere and the applicability regions $\mathcal{A}_i$ are simply sectors on S^1. While Lozano-Pérez

[1] For a moving object and obstacles represented as overlapping unions of convex polyhedra.

(1983) was able to define the form of *C-Space* constraints f_i for $6DOF$, previous work has not been able to formulate the applicability regions in $SO(3)$.

We begin by defining $CO \subseteq \Re^3 \times SO(3)$, the space of forbidden configurations:

$$CO = \{ X \mid \bigvee_a C_a(X) \} \tag{1}$$

where C_a is a *constraint sentence* (see Brooks and Lozano-Peréz (1983)). a is indexed by *C-Space* obstacles. For each *C-Space* obstacle O_a, C_a maps a configuration X to *true* or *false*, depending on whether X is inside O_a. (1) states that if X is inside any *C-Space* obstacle, then it is in CO.

For $X = (x, \Theta)$,

$$C_a(x, \Theta) = \bigwedge_i \Big(\Theta \in \mathcal{A}_i \Rightarrow f_i(x, \Theta) \leq 0 \Big). \tag{2}$$

Let us parse (2). The index i ranges over the set of all C-functions $\{ f_1, \ldots, f_n \}$ which define the *C-Space* obstacle O_a. We call such a set of C-functions a *family* of C-functions. This family is generated by considering pairwise interactions of features on the boundary of A and features on the boundary of B, where A is a convex polyhedron on the moving object, and B is a convex obstacle polyhedron. For a two dimensional example, refer to figures 1.29-32 (chapter 1), which illustrate an obstacle polygon B with four vertices, and a moving polygon A with three vertices. For these two polygons, the family of constraints generated corresponds to all possible interactions of their edges and vertices:

$$\begin{aligned} family_{2D}(A, B) &= \Big(faces(A) \times vert(B) \Big) \bigcup \Big(vert(A) \times faces(B) \Big) \\ &= \Big(\{ e_1, e_2, e_3 \} \times \{ v_1', v_2', v_3', v_4' \} \Big) \bigcup \Big(\{ v_1, v_2, v_3 \} \times \{ e_1', e_2', e_3', e_4' \} \Big). \end{aligned}$$

Each pairing, for example (e_1, v_1'), generates exactly one C-function f_i. In three dimensions, a family of C-functions corresponds to a set of constraints resulting

from the possible interactions of one polyhedral component of the moving object, and one obstacle polyhedron:

$$family_{3D}(A,B) = \Big(faces(A)\times vert(B)\Big)\bigcup\Big(vert(A)\times faces(B)\Big)\bigcup\Big(edges(A)\times edges(B)\Big).$$

Of course, in both two and three dimensions, at a given orientation, only a subset of this family is applicable. For each C-function f_i, there is an associated applicability region $\mathcal{A}_i$. Equation (2) for C_a can be parsed as follows: for a configuration X, for each C-function f_i such that X is in the domain of f_i, $f_i(X)$ must be negative-valued (or zero) for X to be inside the *C-Space* obstacle O_a. To determine whether X is in the domain of f_i, test whether the rotational component of X is within the applicability region $\mathcal{A}_i$.

Next, we define

$$F = \Re^3 \times SO(3) - CO$$

to be the space of free configurations.

Now, for each C-function f_i, $\mathcal{A}_i \subset SO(3)$ is the corresponding portion of rotation space where f_i is applicable. We construct $\mathcal{A}_i$ as the intersection of a set of half-hyperspaces on $SO(3)$:

$$\mathcal{A}_i = \{\,\Theta \in SO(3) \mid \bigwedge_j \Big(g_j(\Theta) \geq 0\Big)\,\} \tag{3}$$

where $g_j : SO(3) \rightarrow \Re$ is an *applicability constraint function* (*ACF*). A C-function f_i is said to be applicable for a configuration $X = (x, \Theta)$ if $\Theta \in \mathcal{A}_i$. In this chapter, we will derive, and prove, the form of the ACFs. Geometrically, the applicability regions $\mathcal{A}_i$ are complicated three dimensional manifolds (with boundary) on the projective 3-sphere. Their boundaries are the two dimensional manifolds $\ker g_j$. (j indexes over the set of functions used to construct $\mathcal{A}_i$. There are typically three or four g_j, as we will see later).

The form of the applicability constraints was heretofore unknown. Many of the representational and algorithmic issues for geometric planning problems with six degrees of freedom rely on a correct formulation of the applicability constraints. With these advances, however, the mathematical framework will be complete, and we can construct the planner of chapter 2 which exploits the geometry.

The work of Brooks and Lozano-Peréz (1983) dealt with surfaces in the *C-Space* $\Re^2 \times S^1$, which are called *C-surfaces*. The obvious extension of this concept for $6DOF$ is a *C-manifold* in $\Re^3 \times SO(3)$. For a C-function f_i we define a *level* C-manifold to be the set of configurations X where f_i is applicable and $f_i(X) = \ell$, for some level ℓ. Thus a level C-manifold is the level set $f_i^{-1}(\ell)$. Of particular interest is the C-manifold

$$\ker f_i = f_i^{-1}(0) = \{ X \mid f_i(X) = 0 \},$$

which contains a boundary patch of a *C-Space* obstacle. Since in the literature, C-manifolds of this form have been called C-surfaces, we shall also employ this term.

We now define paths in *C-Space*. Given a start configuration s and a desired goal configuration g, a successful collision-free path is a continuous function $p : I^1 \rightarrow \Re^3 \times SO(3)$ such that $p(0) = s$, $p(1) = g$, and $p(I^1) \subset F$. I^1 denotes the closed unit interval, $[0, 1]$.

3.3. The Geometric Interpretation for C-functions

Consider the interaction of an obstacle polyhedron B and a moving polyhedron A, where both A and B are convex. Let f_p be in the family of C-functions generated for A and B. f_p models a constraint on the motion of A. For example, f_p might be generated by considering the interaction of a face of A and a vertex of B. For a given orientation Θ, the projection into $\Re^3$ of any (applicable) C-manifold $f_p^{-1}(0)$ is a plane corresponding to a face of the polyhedron resulting from the Minkowski sum of $\bigcirc A$ and B, that is,

$$B \bigcirc A(\Theta) = \{ b + a(\Theta) \mid b \in B, a \in \bigcirc A \}$$

where $a(\Theta)$ denotes vector a rotated to orientation Θ and $\bigcirc A = \{ -a \mid a \in A \}$. (Note that in constructing $\bigcirc A(\Theta)$, the "negation" takes place before the rotation). $B \bigcirc A(\Theta)$ is the projection into $\Re^3$ of the *C-Space* obstacle at orientation Θ. In effect, we have parameterized the plane equations of faces of $B \bigcirc A(\Theta)$ by Θ. Here is the form of the parameterized plane equations derived by Lozano-Pérez (1983): $a_i(\Theta)$ is a vertex of $\bigcirc A(\Theta)$ and b_j is a vertex of B. Recall that the equation of a plane in $\Re^3$ can be expressed as $\{ x \mid \langle N, x \rangle = \langle N, q \rangle \}$, where N is the plane normal and q is a reference point known to be on the plane. Then C-functions take the form:

$$f_p(x, \Theta) = \langle N(\Theta), x \rangle - \langle N(\Theta), (a_i(\Theta) + b_j) \rangle \qquad (4)$$

where x is a point in $\Re^3$. $N(\Theta)$ is the real-space component of the C-manifold normal at orientation Θ, and is defined as follows: for a type (a) C-function, $N(\Theta)$ is the normal of a face of $\bigcirc A(\Theta)$. For a type (b) C-function, $N(\Theta)$ is the normal of a face on B, and hence is constant. For a type (c) C-function, $N(\Theta)$ is the cross-product of an edge on B and an edge on $\bigcirc A(\Theta)$. Furthermore $N(\Theta)$ is normalized to a unit vector when it is non-zero.

The geometric significance of $f_p(x, \Theta)$, is now clear. The value of f_p represents how far the (reference) point x lies above the plane of a face in the Θ-parameterized Minkowski solid. (Assume $(x, \Theta) \in F$). When the projection of x falls on the f_p-face of the Minkowski solid, the metric provided by f_p represents the translational distance to a collision. When the projection falls outside the face, the value of f_p represents the translational distance to the plane of the f_p-face. Hence even though there is no convenient way of talking about distances between configurations in $\Re^3 \times SO(3)$, we can employ the values of C-functions as a metric on the distances of the moving object from obstacles at any configuration. This metric will become important in chapter 6.

3.4. Redundant Constraints

In chapter 2, we gave an informal definition of a redundant constraint (see figure there). We now give the formal definition of a redundant constraint for a configuration $X \in F$. Intuitively, a redundant constraint is one subsumed by nearer, intervening C-functions (lower C-manifolds). Let C denote the set of all applicable, positive-valued C-functions at $X = (x, \Theta)$. For each $f_i \in C$, let s_i be the projection into $\Re^3$ of the kernel of f_i restricted to orientation Θ. That is,

$$s_i = \{ y \in \Re^3 \mid f_i(y, \Theta) = 0 \}.$$

Note that s_i is the projection into $\Re^3$ of the tangent hyperplane at Θ to the level C-manifold for f_i. Intuitively, s_i is the plane of the face of the Minkowski solid determined by f_i, at orientation Θ.

Now, let h_i be the half-space of $\Re^3$ bounded by s_i containing x. Constructing

$$\bigcap_i h_i$$

yields a solid S in $\Re^3$. Those half-spaces bounding S correspond to the non-redundant constraints at X.

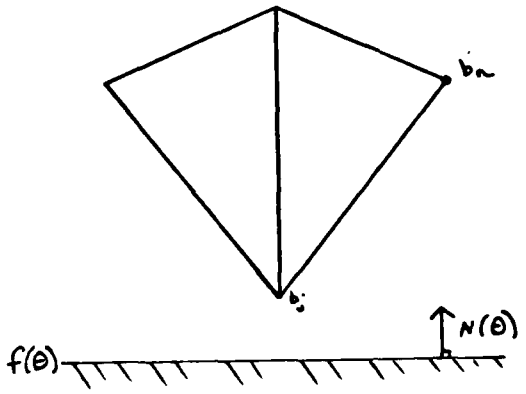


Figure 3.1.

3.5. Applicability Constraints for type (a) and (b) C-functions

We are now in a position to derive the domains of the C-functions. To define the applicability constraints, we consider a family of C-functions in isolation (that is, an environment comprising only the obstacle B and the moving polyhedron A). We perform an analysis to see what generators can interact at what orientations. While C-functions are defined on the "negated object" $\ominus A(\Theta)$, applicability constraints are defined from the "positive object" $A(\Theta)$.

Definition: Consider a constraint c, generated by (g_a, g_b) where the pair (g_a, g_b) is either (a) a face of A and a vertex of B, (b) a vertex of A and a face of B, or (c) an edge of A and an edge of B. We say c is *applicable* at orientation Θ if some pure translation of $A(\Theta)$ can bring $g_a(\Theta)$ in contact with g_b, such that

$$i(A(\Theta)) \bigcap i(B) = \emptyset.$$

See figure 3.1. Let $f(\Theta)$ be a face on a moving polyhedron $A(\Theta)$, with a normal $N(\Theta)$. Let b_j be a vertex on obstacle B. (f, b_j) generates a type (a) constraint. Let R be the set of adjacent vertices of b_j on the edge graph of B, that is,

$$R = \{ b_n \in vert(B) \mid \delta b_n \bigcap \delta b_j \neq \emptyset \}.$$

It is instructive to parse the definition for R. (Recall that δ denotes coboundary). δb_n is the set of edges incident at b_n. If two vertices b_n and b_j have disjoint coboundaries, then they are not adjacent on the edge graph of B. If their coboundaries overlap, then the common element is the edge connecting b_n and b_j.

Theorem III.1: A type (a) constraint generated by (f, b_j) is applicable at orientation Θ if, and only if, for all $b_n \in R$,

$$b_n \cdot N(\Theta) - b_j \cdot N(\Theta) \geq 0. \tag{3.1}$$

If the type (a) constraint is applicable, then (3.1) holds for *all* vertices b_n of B. We will show that considering the vertices in R provides a necessary and sufficient condition for applicability.

Proof: ($\Leftarrow$) Observe that applicability is invariant under translation. We transform the workspace so that the plane of $f(\Theta)$ contains the origin. Then for $x \in \Re^3$, $x \cdot N(\Theta)$ is the perpendicular distance of x from the plane of the constraint. Since face normals are outward-directed, when this distance is positive, then x lies above the plane of $f(\Theta)$. If (3.1) is true, then when b_j is brought to rest on the plane of $f(\Theta)$, then $b_j \cdot N(\Theta) = 0$. Now, for all $b_n \in R$, $b_n \cdot N(\Theta) \geq 0$. Thus all adjacent vertices to b_j are on or above the halfspace boundary. Since A and B are convex, their interiors cannot intersect.

($\Rightarrow$) If we can bring b_j in contact with $f(\Theta)$ while maintaining the disjoint interior criterion, then we have $b_j \cdot N(\Theta) = 0$. No $b_n \in R$ can dip below the surface of $f(\Theta)$, since then the interiors of A and B would intersect. Hence each b_n must lie some distance $d \geq 0$ above the plane of $f(\Theta)$. ∎

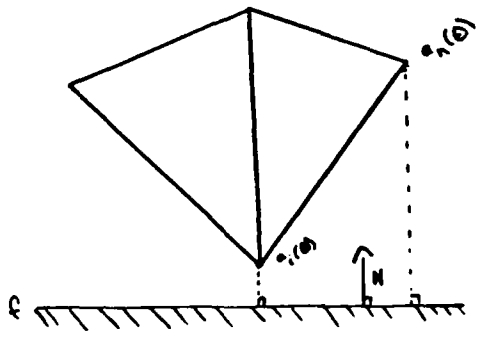


Figure 3.2.

Now, let f be a face of B with normal N. (See figure 3.2). Let a_i be a vertex of A, and

$$R = \{ a_n \in vert(A) \mid \delta a_n \bigcap \delta a_i \neq \emptyset \}$$

be the vertices adjacent to a_i on the edge graph of A.

Theorem III.2: A type (b) constraint generated by (a_i, f) is applicable at orientation Θ if, and only if, for all $a_n \in R$,

$$a_n(\Theta) \cdot N - a_i(\Theta) \cdot N \geq 0. \tag{3.2}$$

Proof: Symmetric case of Theorem (III.1). ∎

Consider

$$g_k(\Theta) = b_n \cdot N(\Theta) - b_j \cdot N(\Theta) \tag{3.3}$$

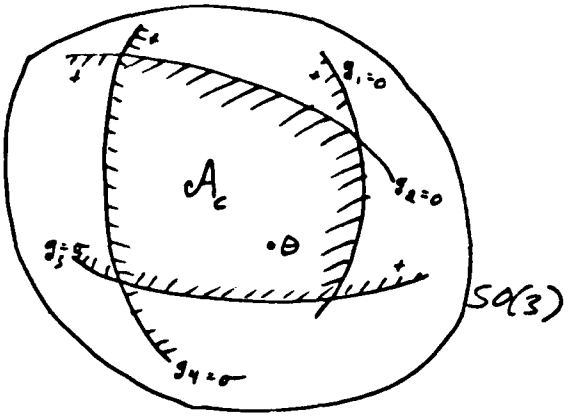


Figure 3.3. The applicability region $\mathcal{A}_c$ is the intersection of the half-hyperspaces where $g_k \geq 0$.

as a mapping $g_k : SO(3) \rightarrow \Re$. We call g_k a type (a) applicability constraint function (ACF). (There are several ACFs for one type (a) C-function—or indeed for any C-function, and they are indexed here by k). For the symmetric case from (3.2), we call

$$g_k(\Theta) = a_n(\Theta) \cdot N - a_i(\Theta) \cdot N \tag{3.4}$$

a type (b) ACF. The region on $SO(3)$ where g_k is positive-valued defines a half-hyperspace of SO(3) (see figure 3.3). (3.2) and (3.1) define the applicability region for a type (a) or (b) constraint as the intersection of these half-hyperspaces. This yields the conjunction promised earlier:

$$\mathcal{A}_c = \{\, \Theta \in SO(3) \mid \bigwedge_k \Big(g_k(\Theta) \geq 0 \Big) \,\}.$$

A C-function c is applicable if and only if for a configuration (x, Θ), each of c's ACFs is positive (or zero) at Θ, that is, $\Theta \in \mathcal{A}_c$. The number of ACFs for a type (a) or (b) constraint is equal to the cardinality of the coboundary of the generating vertex (which is the same as $|R|$).

3.6. Applicability Constraints for Type (c) C-functions

Determining the applicability regions for type (c) C-functions (generated by edge–edge interactions) turns out to be a bit more grueling. We can derive a set of ACFs for type (c) constraints which are analogous to g_k in (3.3) and (3.4). The conjunction of these *type (c) ACFs* is a necessary but not sufficient criterion for applicability. The positive conjunction (the intersection of half-spaces where the type (c) ACFs are positive) forms two, disconnected regions in $SO(3)$. It will become apparent shortly how these regions arise, but let us pause, before bringing in some complicated machinery, to survey their topology. In one region $\mathcal{A}$ the type (c) constraint is applicable, in the other $\mathcal{A}'$, it is not. To determine which region Θ is in, we use a set of related functions termed *disambiguating applicability constraints (DACs)*. Fortunately, the symmetric region $\mathcal{A}'$ where the ACFs are positive but the constraint is not applicable is disconnected from the valid applicability region $\mathcal{A}$ (where the ACFs are positive and the constraint *is* applicable) by a region $\overline{\mathcal{A}}$, where the ACFs do not hold (see figure 3.4). We will demonstrate that since $\mathcal{A}$ is disconnected from $\mathcal{A}'$, it is possible to plan continuous paths within $\mathcal{A}$ with heed only for the basic type (c) ACFs. Both type (c) ACFs and DACs are functions of the form $g : SO(3) \rightarrow \Re$; however, they are considerably more complicated than (3.3) and (3.4), above.

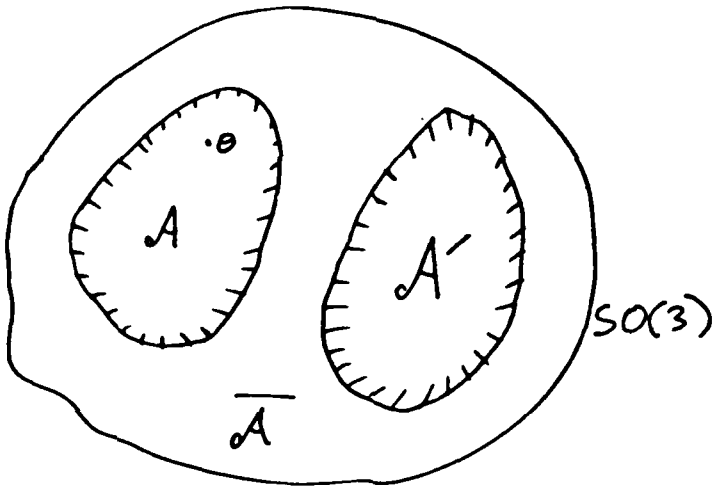


Figure 3.4. The space $SO(3)$ showing A, A', and $\overline{A}$. The type (c) ACFs determine whether Θ is in the set $\overline{A}$ or in $A \bigcup A'$. The DACs determine, for $\Theta \in A \bigcup A'$, whether Θ is in A or A'.

3.6.1. The Basic ACFs for Type (c) Constraints

Let c be a type (c) constraint generated by the pair of edges (e_a, e_b). As usual $e_a(\Theta)$ denotes e_a rotated to orientation Θ. We will define type (c) ACFs which provide a necessary criterion for applicability. In conjunction with the DACs (below), the type (c) ACFs form a complete characterization of the applicability of type (c) C-functions. We employ the following construction: imagine trying to make the midpoints of e_a and e_b touch while maintaining the disjoint interior criterion for A and B. We then allow A to pivot about

$$v = \text{mid}(e_b) = \text{mid}(e_a(\Theta)) \tag{3.5}$$

while maintaining disjoint interiors. Keeping (3.5), for what orientations (values of Θ) are *the interiors of A and B disjoint*?

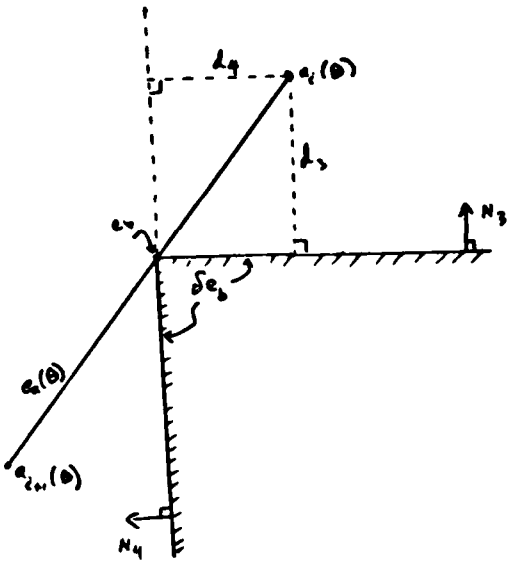


Figure 3.5. A section view through e_b (e_b is orthogonal to the page).

Let us denote the vertices of e_a and e_b as follows: (see figures 3.5 and 3.6) $vert(e_a) = (a_i, a_{i+1})$ and $vert(e_b) = (b_j, b_{j+1})$. Now, e_a bounds 2 faces f_1 and f_2 on A; let their normals be N_1 and N_2. Similarly, let the normals for the faces f_3 and f_4 cobounding e_b be N_3 and N_4.

Theorem III.3: If a type (c) constraint generated by (e_a, e_b) is applicable at orientation Θ, then

$$-d_1(\Theta)d_2(\Theta) \geq 0 \tag{3.6}$$

and

$$-d_3(\Theta)d_4(\Theta) \geq 0 \tag{3.7}$$

where

$$d_1(\Theta) = b_j \cdot N_1(\Theta) - \text{mid}(e_b) \cdot N_1(\Theta) \tag{3.6a}$$

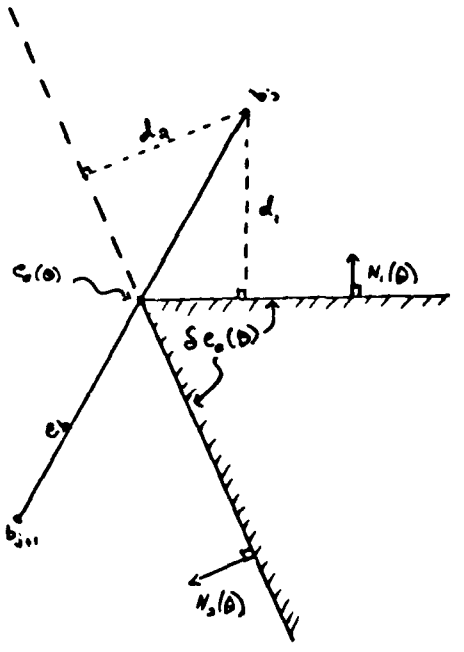


Figure 3.6. A section view through $e_a(\Theta)$ ($e_a(\Theta)$ is orthogonal to the page).

$$d_2(\Theta) = b_j \cdot N_2(\Theta) - \text{mid}(e_b) \cdot N_2(\Theta) \tag{3.6b}$$

$$d_3(\Theta) = a_i(\Theta) \cdot N_3 - \text{mid}(e_a(\Theta)) \cdot N_3 \tag{3.7a}$$

$$d_4(\Theta) = a_i(\Theta) \cdot N_4 - \text{mid}(e_a(\Theta)) \cdot N_4 \tag{3.7b}$$

(We express (3.6-7) in this form rather than as $d_1(\Theta)d_2(\Theta) \leq 0$ in order to preserve the positive sign convention for all the ACFs).

Proof: Refer to figures 3.5 and 3.6. Again, since applicability is invariant under translation, we transform the workspace so that $\text{mid}(e_b)$ is at the origin. With $\text{mid}(e_a(\Theta))$ fixed at $\text{mid}(e_b)$, $d_i(\Theta)$ for $i = 1, 2$ is the distance of b_j above the plane of f_i; for $i = 3, 4$, this is the distance of $a_i(\Theta)$ above the plane of f_i. We allow e_a to rotate about $v = \text{mid}(e_b)$ with 3 degrees of freedom. Observe that a_i and a_{i+1} may not dip below the surface of B, and that b_j and b_{j+1} may not fall below the surface of A. This is clearly enforced by considering only the planes of the

faces cobounding e_a and e_b. If the type (c) constraint is applicable at Θ, then $d_1(\Theta)$ and $d_2(\Theta)$ can never both be positive, nor both negative, for in these cases e_b will intersect the interior of A. We see this as follows: If $d_1(\Theta) < 0$ and $d_2(\Theta) < 0$, then b_j is inside both halfspaces, and some point on the line segment $(\text{mid}(e_b), b_j)$ must be inside A. If $d_1(\Theta) > 0$ and $d_2(\Theta) > 0$, then b_{j+1} is inside both half spaces, and some point on the line segment $(\text{mid}(e_b), b_{j+1})$ must be inside A.

Hence $d_1(\Theta)d_2(\Theta) \leq 0$. This immediately yields (3.6). A similar and symmetric argument yields (3.7). ∎

3.7. Disambiguating Applicability Constraints (DACS) for Type (c) Constraints

The basic type (c) ACFs take into account edge-edge interactions, but do not model the interactions of the faces they bound. In order to preserve the disjoint interior criterion, we introduce Disambiguating Applicability Constraints (DACs) as follows. DACs are constraints on the tangent vectors to faces cobounding e_b and e_a; assuming that the basic ACFs have determined that $\Theta \in \mathcal{A} \bigcup \mathcal{A}'$, DACs discriminate between $\mathcal{A}$ and $\mathcal{A}'$. In fact, the DACs are necessary and sufficient conditions for applicability. We split the type (c) applicability computations between the basic type (c) ACFs and the DACs for reasons relating to the algebra system, which is described in chapter 4. Our proofs draws heavily on constructions employing a separating plane.

The Separating Plane Construction

Join the midpoints of $e_a(\Theta)$ and e_b together as usual. Consider the plane P containing $v = \text{mid}(e_a(\Theta)) = \text{mid}(e_b)$, whose normal is $e_a(\Theta) \times e_b$. Assume without loss of generality that $e_a(\Theta) \times e_b \neq \mathbf{0}$. P contains both e_b and $e_a(\Theta)$. Suppose that the type (c) ACFs for constraint c are positive-valued (or zero), i.e., (3.6) and (3.7) hold. Hence each vertex of e_b is on or above the plane of one face cobounding $e_a(\Theta)$, and each vertex of $e_a(\Theta)$ is on or above the plane of one face cobounding e_b. Refer to figures 3.6 and 3.5 once more. By reason of the ACF values for c and the convexity of A and B, some open halfspace P_B of $\Re^3$ which is bounded by P must contain $i(B)$ entirely, and some open halfspace P_A bounded by P must contain $i(A(\Theta))$ entirely:

$$B \subset \kappa(P_B)$$

$$A(\Theta) \subset \kappa(P_A).$$

(Recall that $\kappa(S)$ denotes the *closure* of a set S: $\kappa(S) = i(S) \bigcup \partial S$).

Now, if c is not applicable, then $i(A(\Theta)) \bigcap i(B) \neq \emptyset$. This means that $A \subset \kappa(P_B)$ also, since unless $P_A = P_B$, then P would separate $i(A(\Theta))$ from $i(B)$. We conclude

that for all $\Theta' \in \mathcal{A}'$, $i(A(\Theta')) \subset P_B$ and $i(B) \subset P_B$. By a symmetric argument, for all $\Theta \in \mathcal{A}$, plane P separates $i(A(\Theta))$ from $i(B)$. To summarize: If the constraint c generated by (e_a, e_b) is applicable at orientation Θ, then $i(A(\Theta)) \bigcap i(B) = \emptyset$. Therefore there exists a separating plane between $i(A(\Theta))$ and $i(B)$. On the other hand, if c is not applicable, there exists no such separating plane, for then the interiors could not intersect. Furthermore, if $e_a(\Theta) \times e_b \neq 0$, then there exists exactly one separating plane that contains all four points $vert(e_a(\Theta)) \bigcup vert(e_b)$. We formalize these results in the following lemma:

In this lemma, we abbreviate $e_a(\Theta)$ by e_a, and $A(\Theta)$ by A.

Lemma III.4.1: *(Existence and uniqueness of the separating plane).* Join together the midpoints of e_a and e_b. Assume that $e_a \times e_b \neq 0$. The constraint c generated by (e_a, e_b) is applicable if, and only if, the plane P containing e_a and e_b separates the interior of A from the interior of B.

Proof: ($\Leftarrow$) If P separates $i(A)$ from $i(B)$, then $i(A) \bigcap i(B) = \emptyset$. Therefore the constraint c is applicable. ∎

Proof: ($\Rightarrow$) If c is applicable, then there exists exactly one separating plane between $i(A)$ and $i(B)$, and this plane is P. To see this, first observe that if $i(A) \bigcap i(B) = \emptyset$, then by convexity there must exist some separating plane. Assume that this plane does not have normal $e_a \times e_b$. In this case, the plane cannot contain both e_a and e_b. Since the plane contains the midpoints of both edges, it must intersect either e_a or e_b in a non-parallel cut. But in this case, the plane intersects the interior of either A or B. Thus it cannot be a separating plane. Since there must exist some separating plane, it must have normal $e_a \times e_b$. ∎

The strategy for defining DACs is as follows. For each face cobounding e_b, we choose a point in the interior of that face. The basic type (c) ACFs ensure that e_b is outside the interior of $A(\Theta)$, and that $e_a(\Theta)$ is outside the interior of B; the DACs ensure that the faces cobounding e_b lie on the opposite side of P from the the faces cobounding $e_a(\Theta)$. If the type (c) constraint is applicable, then $i(A(\Theta))$ must lie in a half-space bounded by P complementary to the half-space bounded by P containing $i(B)$. The DACs ensure that if the faces cobounding $e_a(\Theta)$ lie in

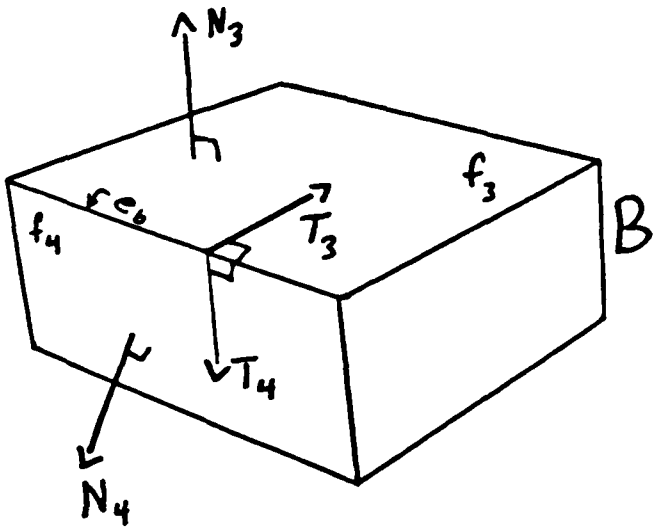


Figure 3.7. The Tangent Vectors T_3 and T_4 to the faces cobounding e_b.

$\kappa(P_A)$, then the faces cobounding e_b must lie in $\kappa(P_B)$, with $P_A \neq P_B$. Since A and B are convex, this suffices to show that A and B lie in complementary half-spaces bounded by P.

The vertices of $e_a(\Theta)$ and e_b lie on P. Let p_1, p_2 be points in the interior of the faces cobounding e_a, and p_3, p_4 be points in the interior of the faces cobounding e_b. The DACs ensure that p_1 and p_2 lie on one side of P, and that p_3 and p_4 lie on the other.

The points inside the faces cobounding e_b and e_a are chosen as follows. For each edge e on B and A, we construct a pair of tangent vectors, (T_1, T_2), where T_1 and T_2 are tangent and interior to the faces cobounding e. T_1 and T_2 are also perpendicular to e. For an edge e_a on A, $(T_1(\Theta), T_2(\Theta))$ will clearly rotate with e_a and A, maintaining these criteria. The *tangent pair* for e_b is shown in figure 3.7. Formally, we proceed as follows:

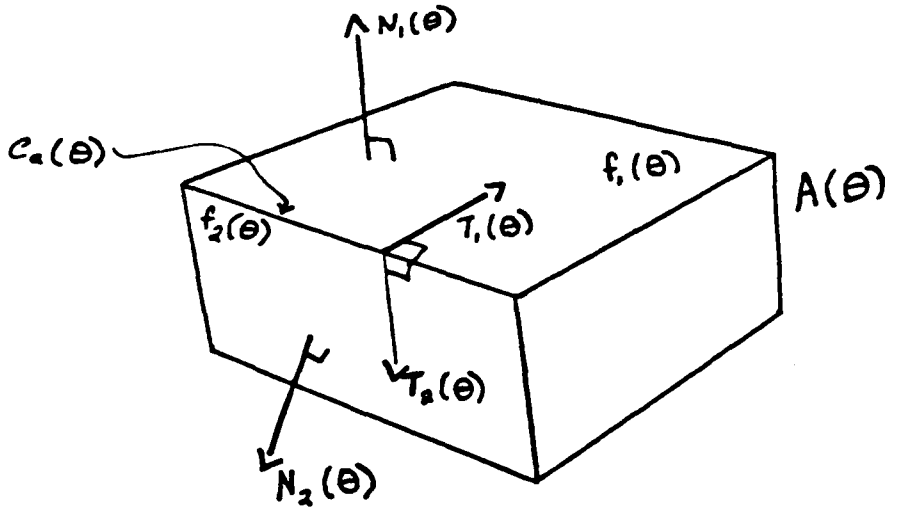


Figure 3.8. The tangent pairs and normals for $e_a(\Theta)$.

Definition: A *tangent vector* to $\Re^3$ (O'Neill (1966)) is a pair $(v, p) \in \Re^3 \times \Re^3$, interpreted as the vector v applied to point p. We will sometimes write v_p for (v, p), or, when there is no ambiguity about the point of application, we simply write v.

Definition: Consider an edge e on a polyhedron P. Let f_1, f_2 be the faces that cobound e, and let N_1, N_2 be their normals. A *tangent pair* for e is a pair of tangent vectors to $\Re^3$, (T_1, T_2), both applied to mid(e). T_i is perpendicular to e and to N_i, and it is directed into the interior of f_i when applied to mid(e) ($i = 1, 2$). In other words,

$$T_i = k(N_i \times e) \qquad (i = 1, 2)$$

where $k \in \{+1, -1\}$ is chosen to orient T_i into the interior of f_i. $N_i \times e$ indicates the cross product of N_i and the directed edge vector for e.

Refer to figures 3.7 and 3.8. We will now construct DACs. Let (T_3, T_4) be the tangent pair for e_b, and let N_3, N_4 be the normals to the faces cobounding e_b. Let $(T_1(\Theta), T_2(\Theta))$ be the tangent pair for $e_a(\Theta)$, and let $N_1(\Theta)$, $N_2(\Theta)$ be the normals to the faces cobounding $e_a(\Theta)$. Thus $T_i \cdot N_i = 0$ (for $i = 1, 2, 3, 4$). Keeping with this numbering convention, let f_i be the face with normal N_i. As usual, we imagine joining together the midpoints of e_b and $e_a(\Theta)$.

Let $N_P(\Theta)$ be the normal to the plane P, that is, $N_P(\Theta) = e_a(\Theta) \times e_b$. Assume without loss of generality that $N_P(\Theta) \neq 0$. The plane containing $\text{mid}(e_a(\Theta)) = \text{mid}(e_b)$ with normal $N_P(\Theta)$ also contains $e_a(\Theta)$ and e_b. We construct DACs which ensure that $i(A(\Theta))$ is on one side of P, and that $i(B)$ is on the other side. To ensure that the points $\text{mid}(e_b) + T_3$ and $\text{mid}(e_b) + T_4$ lie on the same side of P, we have the constraint

$$\text{sign}(T_3 \cdot N_P(\Theta)) = \text{sign}(T_4 \cdot N_P(\Theta))$$

which may be written

$$(T_3 \cdot N_P(\Theta))(T_4 \cdot N_P(\Theta)) > 0.$$

Assume without loss of generality that the signs are non-zero. The case where one sign is zero is easily handled by examining the other sign. To ensure that the points $\text{mid}(e_a(\Theta)) + T_1(\Theta)$ and $\text{mid}(e_a(\Theta)) + T_2(\Theta)$ lie on the same side of P, we have the symmetric constraint

$$\text{sign}(T_1(\Theta) \cdot N_P(\Theta)) = \text{sign}(T_2(\Theta) \cdot N_P(\Theta))$$

Now, we must ensure that the two half-spaces are complementary. This is enforced by insisting that the signs are opposite. All of the following must be true:

$$\begin{aligned} k_B &= \text{sign}(T_3 \cdot N_P(\Theta)) \\ &= \text{sign}(T_4 \cdot N_P(\Theta)) && (3.8a) \\ k_A &= \text{sign}(T_1(\Theta) \cdot N_P(\Theta)) \\ &= \text{sign}(T_2(\Theta) \cdot N_P(\Theta)) && (3.8b) \\ k_A &\neq k_B && (3.8c) \end{aligned}$$

Equations (3.8a-c) embody the DACs we require.

Theorem III.4: Let c be a type (c) C-function generated by (e_a, e_b). Assume the tangent pairs for e_b and e_a, and normals to the faces cobounding e_b and e_a are as above. Then c is applicable if, and only if, the all the DACs (3.8a-c) hold.

Proof: ($\Rightarrow$) Assume the type (c) constraint is applicable, but that at least one of (3.8a-c) is false. We will demonstrate a contradiction. Join the midpoints of $e_a(\Theta)$ and e_b, as usual. If any of the DACs is false, then P does not separate $i(A(\Theta))$ from $i(B)$: a contradiction. ∎

Proof: ($\Leftarrow$) We show that if the DACs hold, then c is applicable: if these conditions are true, then P is a separating plane. Therefore the interiors cannot intersect, and c is applicable. ∎

3.8. On the Structure of the Type (c) Applicability Regions on $SO(3)$

In this section, we prove a theorem on the structure of the regions $\mathcal{A}$, $\mathcal{A}'$, and $\bar{\mathcal{A}}$ for type (c) constraints, (see figure 3.4) which yields an immediate completeness result for our formulation of ACFs and DACs. As promised, we will show that $\mathcal{A}$ and $\mathcal{A}'$ are disconnected on $SO(3)$, and that the region $\bar{\mathcal{A}}$ separates them. Our proof draws heavily on constructions employing a separating plane (lemma III.4.1).

Theorem III.5: $\bar{\mathcal{A}}$ disconnects $\mathcal{A}$ from $\mathcal{A}'$ on $SO(3)$.

Proof: We first observe that by definition,

$$\bar{\mathcal{A}} \bigcup (\mathcal{A} \bigcup \mathcal{A}') = SO(3)$$

(see (3.6), (3.7) for confirmation). Recall the separating plane construction: we saw that for all $\Theta' \in \mathcal{A}'$, $i(A(\Theta')) \subset P_B$ and $i(B) \subset P_B$. Let $\bar{P}_B$ denote the interior of the complement of P_B: $\bar{P}_B = i(\Re^3 - P_B)$. By a symmetric argument, for all $\Theta \in \mathcal{A}$, plane P separates $i(A(\Theta))$ from $i(B)$. If $\mathcal{A} \cup \mathcal{A}'$ is path-connected, then there exists a continuous function, $p : I^1 \rightarrow SO(3)$, such that $p(0) = \Theta$, $p(1) = \Theta'$, and $p(I^1) \subset \mathcal{A} \cup \mathcal{A}'$. Furthermore, if $\mathcal{A} \cup \mathcal{A}'$ is path-connected, then for all $t \in I^1$, either $i(A(p(t))) \subset P_B$, or $i(A(p(t))) \subset \bar{P}_B$ (assume without loss of generality that for all t, $e_b \times e_a(p(t)) \neq 0$). Note that for all t,

$$P \bigcap i(A(p(t))) = \emptyset.$$

Hence in traversing the path p in rotation space, A is required to "flip" over P from P_B to $\bar{P}_B$, without its interior ever intersecting P. This is clearly impossible if continuity is to be preserved. ∎

3.9. Orienting Type (c) Constraints

Consider affixing $\text{mid}(e_a(\Theta))$ to $v = \text{mid}(e_b)$ as usual. Refer once more to 3.5 and 3.6. The cross product

$$N_P(\Theta) = e_a(\Theta) \times e_b$$

when applied to v will for some Θ point out of P_B and into P_A; for other Θ, $N_P(\Theta)$ will point into P_B and out of P_A. (Assume for now that $N_P(\Theta) \neq 0$.) Hence for some orientations $N_P(\Theta)$ is the correct (unnormalized) real-space normal for constraint (e_a, e_b); for other orientations we must employ $-N_P(\Theta)$. When applied to v, the real-space normal $kN_P(\Theta)$ (for $k \in \{+1, -1\}$) must always point out of P_B and into P_A. The following rule for choosing k is stated without proof:

$$k = \text{sign}(N_P(\Theta) \cdot I(\Theta)) \tag{3.12}$$

where $I(\Theta) = T_1(\Theta) + T_2(\Theta)$.

However, it is easy to see that we need not compute this dot product each time we use the C-function. k (and the orientation of $N_P(\Theta)$) will be invariant in regions of $\mathcal{A}$ where the signs of the ACFs are invariant. For example, if k is positive for some $\Theta \in \mathcal{A}$ and

$$d_3(\Theta) > 0 \text{ and } d_4(\Theta) < 0, \tag{3.13}$$

then clearly *wherever* (3.13) holds, then k must be positive. Also, wherever

$$d_3(\Theta) < 0 \text{ and } d_4(\Theta) > 0,$$

k must be negative. This argument should be quite obvious if the reader imagines how the cross product of the edges changes as e_a pivots about $\text{mid}(e_b)$. This leads to the following simple algorithm for orienting a type (c) C-function c. Essentially, we can just compute (3.12) once, and record the signs of the ACFs at that orientation.

(i) For some Θ, compute the values of $d_i(\Theta)$ $(i = 1, 2, 3, 4)$ for the type (c) ACFs. If c is not applicable, then stop.

(ii) If k has not been computed yet, calculate k as in (3.12). (Assume $k \neq 0$). Record the signs of $d_3(\Theta)$ and $d_4(\Theta)$ for c. We call this pair of signs the *sign map* for c.

(iii) If a k and sign map have been computed for c, then compare the recorded sign map to the current sign map for $d_3(\Theta)$ and $d_4(\Theta)$. If the sign maps are equal, use k to orient c; otherwise use $-k$.

3.10. Singularities and Special Cases

Our analysis of type (c) ACFs and DACs assumes that $e_a(\Theta)$ and e_b are never aligned, i.e., that their cross-product is never zero. In addition, our algorithm for orienting type (c) C-functions assumes that no function d_i is zero. The cross product will be zero when $e_a(\Theta)$ is parallel to e_b, and an ACF will be zero when either $e_a(\Theta)$ is aligned with a face cobounding e_b, or when e_b is aligned with a face cobounding $e_a(\Theta)$. In practice, these special cases will arise frequently. Fortunately, they can be ignored. Consider the following: The vertices of e_a generate type (b) constraints with the faces cobounding e_b; and the vertices of e_b generate type (a) constraints with the faces cobounding e_a. In the cases where $e_a(\Theta)$ is aligned with e_b or a face cobounding e_b (or in the symmetric case), some of these constraints will also be applicable. In these aligned cases we say that the type (c) constraint is *subsumed* by the neighboring type (a) and (b) constraints, because the disjoint interior criterion will be enforced by the type (a) and (b) constraints alone. This can be seen as follows: (see figure 3.5) suppose some ACF, for example d_3, is zero-valued at Θ, and that $\Theta \in \mathcal{A}$. Then both $\mathrm{mid}(e_a(\Theta))$ and $a_i(\Theta)$ can be brought to rest on the plane of f_3, while preserving the disjoint interior criterion. Since $a_{i+1}(\Theta)$ is also lies on $e_a(\Theta)$, it too may be brought to rest on the plane of f_3. Clearly, the type (b) constraints generated by (a_i, f_3) and (a_{i+1}, f_3) must also be applicable at orientation Θ. At this aligned orientation, the type (c) constraint ensures the following: *while* $\mathrm{mid}(e_a(\Theta))$ *is on the plane of* f_3, $a_i(\Theta)$) *must also lie on the plane of* f_3. This is precisely the condition enforced by the equivalent pair of type (b) constraints. Symmetric arguments hold for the other ACFs.

3.11. Level ACFs

For ACFs, there is an analogous concept to a level C-Manifold. Let $g : SO(3) \rightarrow \Re$ be an ACF for a C-function c. An *ACF Boundary* is the space of rotations where c is applicable and g is zero:

$$\ker g = \{\, \Theta \in SO(3) \mid g(\Theta) = 0 \,\}.$$

A *Level ACF* is the space of rotations where c is applicable and g is some constant

value ℓ:

$$\{\Theta \in SO(3) \mid g(\Theta) = \ell\}.$$

Recall the geometric interpretation for ACFs. Consider a type (b) constraint (see figure 3.5). A path $p : I^1 \rightarrow SO(3)$ along a level ACF for the constraint (a_i, f_3) would, if the midpoints of the edges were affixed, preserve $a_i(\Theta)$ at a constant height above the plane of f_3.

3.12. A Note on the Computation and Algebra of Applicability Constraints

The implemented planning system contains an algebra system (described in chapter 4), which performs the computations relevant to the applicability constraints. We would like to make the computation as simple as possible, for otherwise an implementation might be infeasible.

We have shown that there are four types of computations for the applicability constraints:

(i) Type (a) ACFs (3.3) which determine the applicability of type (a) C-functions.

(ii) Type (b) ACFs (3.4) which determine the applicability of type (b) C-functions.

(iii) [Basic] Type (c) ACFs (3.6) and (3.7), which provide a necessary but not sufficient condition for the applicability of type (c) C-functions.

(iv) DACs (3.8a-c) which provide necessary and sufficient conditions for type (c) applicability.

However, it is not hard to show that the real-valued functions for (iii) and (iv) are composed of simple type (a) and (b) ACFs. We will demonstrate this as follows. Let $\mathcal{N}^3$ be the space of normals to planes in $\Re^3$. Note that $\mathcal{N}^3$ is of course isomorphic to $\Re^3$. We now define the functions F_A and F_B to model the computation of type (a) and (b) ACFs. These functions will be composed to compute the more complex type (c) ACFs and DACs. Let F_A and F_B be real-valued functions

$$F_A, F_B : \Re^3 \times \Re^3 \times \mathcal{N}^3 \rightarrow \Re$$

where

$$F_A(b_n, b_j, N, \Theta) = b_n \cdot N(\Theta) - b_j \cdot N(\Theta)$$

and

$$F_B(a_n, a_i, N, \Theta) = a_n(\Theta) \cdot N - a_i(\Theta) \cdot N.$$

Clearly, F_A and F_B can be used to compute ACFs for all type (a) and (b) constraints. They can also be used to compute type (c) ACFs as follows:

$$d_i(\Theta) = \begin{cases} F_A(b_j, \text{mid}(e_b), N_i, \Theta), & \text{if } i = 1, 2; \\ F_B(a_i, \text{mid}(e_a), N_i, \Theta), & \text{if } i = 3, 4. \end{cases}$$

$N_P(\Theta)$ is already computed as the real-space normal for a type (c) C-function. With $N_P(\Theta)$ in hand, DACs can be computed using F_B and F_A. This is because DACs are essentially constraints on tangent vectors to the faces of the polyhedra in question, and the tangent space of $\Re^3$ is isomorphic to its normal space. We will show how to compute DACs using type (a) and (b) ACFs. Our trick for rotating a tangent vector (v, p) simply involves rotating the line segment $(p, v + p)$ to $(p(\Theta), [v + p](\Theta))$. For example,

$$T_1(\Theta) \cdot N_P(\Theta) = F_B\Big(\text{mid}(e_a(\Theta_0)) + T_1(\Theta_0), \text{mid}(e_a(\Theta_0)), N_P(\Theta), \Theta\Big)$$

Here Θ_0 denotes some fixed orientation. Typically Θ_0 is the identity element for the rotation group, i.e., it denotes no rotation at all, and will be the orientation in which the polyhedra are given, and in which the tangent pairs are initially computed. In particular, $[T_i(\Theta_0)](\Theta) = T_i(\Theta)$.

Our reduction of all applicability computation to a few simple functions is partially motivated by æsthetics, and partially by the design of an algebra system for our planner. The reduction will admit a simpler and more elegant design.

3.12.1. A Conjecture

Let us make one final comment on type (c) ACFs. For each type (c) C-function, there are two type (c) ACFs. One type (c) ACF (3.6) is the product of two type (a) ACFs; and the other (3.7) is the product of two type (b) ACFs. These products are constrained to be negative. In practice, we would probably wish only to compute the value of each subresult (d_i) for each type (a) and (b) ACF, and then compute a logical conjunction to determine when one is negative and the other

positive, instead of computing their product.[2] We conjecture that the composition of type (c) C-functions and ACFs reflects the underlying algebraic structure of these constraints: observe that each type (c) face $f_{a,b}$ of the Minkowski solid $B \ominus A(\Theta)$ is the composition (by direct sum) of an edge on A and an edge on B:[3]

$$f_{a,b} = e_a(\Theta) \bigoplus e_b. \tag{3.14}$$

Similarly, the (real-space) normal $N_{a,b}$ to such a face is the composition (by vector cross-product) of an edge on A and and edge on B:

$$N_{a,b} = e_a(\Theta) \times e_b. \tag{3.15}$$

In this chapter we have derived a new symmetry, a symmetry for the ACFs of type (c) constraints. In particular, it is now clear that type (c) ACFs are the composition (by scalar multiplication) of a pair of type (a) or (b) ACFs.

[2] This approach is taken for the implemented planner.

[3] Equations (3.14) and (3.15) are from Lozano-Pérez (1983).

4
Mathematical Tools for Motion Planning in a Six Dimensional Configuration Space

4.1. Introduction

Our earlier presentation of representational issues and applicability constraints in $\Re^3 \times SO(3)$ addressed basic theoretical issues for the motion planning problem. In this chapter we discuss specific issues which were critical for the implementation of the planning system described in chapter 2. The fundamental issue is the *intersection problem* in high-dimensional configuration spaces:

(i) How do we intersect high-dimensional level C-Manifolds to construct an intersection manifold?

(ii) How do we intersect a trajectory in configuration space with *C-Space* constraints?

Examples and applications of these results may be found in chapters 1 and 2.

We will proceed as follows. First, as a "simple" example, we will solve these problems for the configuration space $\Re^2 \times S^1$. For this space the algebra is not unreasonable and illustrates some of the complexities of planning for the $6DOF$ case. However, in $\Re^3 \times SO(3)$, the equations for some constraints (notably, type (c) constraints) can fill several pages. For this reason, I first computed the general form of the intersections for an arbitrary constraint, and then solved all intersections using `Macsyma` (LCS (1983)). The results were then optimized and compiled into `Lisp`. For all practical purposes these results are in machine readable form only. For

example, using Euler Angles parameterized by $\Theta = (\psi, \theta, \phi)$ for three-dimensional rotations,[1] a type (b) constraint in Macsyma becomes:

```
((-XC(AI)*XC(NGJ)*COS(PHI)-XC(AI)*YC(NGJ)*SIN(PHI))*COS(THETA)
 +XC(AI)*ZC(NGJ)*SIN(THETA)-YC(AI)*YC(NGJ)*COS(PHI)
 +YC(AI)*XC(NGJ)*SIN(PHI))
 *COS(PSI)
 +((YC(AI)*XC(NGJ)*COS(PHI)+YC(AI)*YC(NGJ)*SIN(PHI))*COS(THETA)
  -YC(AI)*ZC(NGJ)*SIN(THETA)-XC(AI)*YC(NGJ)*COS(PHI)
  +XC(AI)*XC(NGJ)*SIN(PHI))
  *SIN(PSI)-ZC(AI)*ZC(NGJ)*COS(THETA)
 +(-ZC(AI)*XC(NGJ)*COS(PHI)-ZC(AI)*YC(NGJ)*SIN(PHI))*SIN(THETA)
 -YC(NGJ)*COS(PHI)+XC(NGJ)*SIN(PHI)+ZC(NGJ)*Z+YC(NGJ)*Y+XC(NGJ)*X
 -ZC(BJ)*ZC(NGJ)-YC(BJ)*YC(NGJ)-XC(BJ)*XC(NGJ).
```

This is the simplest of the constraints; a type (c) constraint is over 10 times as long. For $\Re^3 \times SO(3)$ our approach has been to (1) derive these constraints (and the ACFs) from some arbitrary representation for rotations, (2) reduce each constraint to a series of simpler, canonical forms which are linear, bilinear, or quadratic in the terms of interest, and (3) develop simple mathematical procedures for operating on the canonical forms.

For example, to construct an intersection manifold for n constraints, we essentially need to solve a set of n simultaneous equations, each of the form

$$f(X) = 0. \qquad (X \in \Re^3 \times SO(3))$$

We proceed as follows. Let $D = \{ x, y, z, \psi, \theta, \phi \}$ be the set of all the degrees of freedom. First we select P, a subset of $6 - n$ elements of D. P will parameterize the intersection manifold. The variables in P will be the free variables which the planner can choose; the variables $D - P$ will vary dependently with P so as to stay on the intersection manifold. Mechanically, this entails (1) solving the n constraints simultaneously eliminating all but one variable in $D - P$, and (2) expressing all dependent degrees of freedom $D - P$ in terms of the free variables P.

The canonical forms are expressions for C-functions which make explicit the coefficients of the dependent variables $(D - P)$ themselves, and of the sines and cosines of these variables. 13 complicated equations describe the canonical forms

[1] Euler angles are implemented as rotation matrices in the planner. See Symon (1971).

of a C-function, and 9 equations are needed for a type (a) or (b) ACF.[2] Complete Macsyma listings of these procedures are provided in an appendix. Before wading into these waters, however, let us turn our attention to the configuration space $\Re^2 \times S^1$.

We will adhere to the definitions and conventions established in chapter 3.

4.2. The Intersection Problem in $\Re^2 \times S^1$

The find-space and find-path problems in $\Re^2 \times S^1$ are of considerable intrinsic interest. We have suggested that good algorithms for the two dimensional Movers' problem could be developed by planning along the intersections of constraints. Some of the necessary theoretical tools for this approach are presented in this section. These results illustrate the principles necessary for planning along intersection manifolds in $\Re^3 \times SO(3)$. The derivations are simpler because (1) the constraints are simpler and (2) the applicability regions are merely sectors on the unit circle. A complete, general path planner has been implemented for this problem (see Brooks and Lozano-Peréz (1983)). This section serves both as a pedagogic example and as a presentation of a new approach to the planning problem in $\Re^2 \times S^1$.

To plan paths along the intersections of constraints, we must be able to construct the intersection manifold of some set of constraints. To preserve tradition (see Brooks and Lozano-Peréz (1983), for example), we will call any level-0 C-manifold a *C-surface*. A C-surface is the space of configurations where a C-function is applicable and zero-valued. C-surfaces are interesting because they bound *C-Space* obstacles. We will derive the form of the intersection of any two C-surfaces in $\Re^2 \times S^1$. Each C-surface is a 2-dimensional manifold in $\Re^2 \times S^1$, and their intersection manifold is a curve p in $\Re^2 \times S^1$. We derive a curve p which is parametric in θ.[3] Since there are 2 types of C-surfaces (type (a) and (b)), there are 3 types of intersection manifolds.

4.2.1. The Intersection of Two C-Surfaces in $\Re^2 \times S^1$

We describe a technique for finding the intersection of two C-surfaces for the two dimensional mover's problem with rotations. Throughout this discussion of

[2] We see now why it was desirable to express all ACFs and DACs as compositions of type (a) and (b) ACFs.

[3] Recall that (x, y, θ) is a typical point in the *C-Space* $\Re^2 \times S^1$.

$\Re^2 \times S^1$, we will employ the abbreviations $C = \cos\theta$ and $S = \sin\theta$. The surfaces are embedded in a 4-dimensional manifold and expressed as functions on (x, y, C, S) with the added constraint that $C^2 + S^2 = 1$. A system of equations for two surfaces can then be solved for x and y in terms of C and S.

Two type (a) constraint surfaces are functions of the form $f(x, y, \theta) = 0$, for example:

$$\sin(\theta + \lambda_i)y + \cos(\theta + \lambda_i)x - \|b_j\| \cos(\theta + \lambda_i - \gamma_j) - \|a_i\| \cos(\lambda_i - \eta_i) \qquad (a1)$$

$$\sin(\theta + \lambda_i')y + \cos(\theta + \lambda_i')x - \|b_j'\| \cos(\theta + \lambda_i' - \gamma_j') - \|a_i'\| \cos(\lambda_i' - \eta_i') \qquad (a2)$$

Similarly, two type (b) surfaces are:

$$\sin(\phi_j)y + \cos(\phi_j)x - \|a_i\| \cos(\theta - \phi_j + \eta_i) - \|b_j\| \cos(\phi_j - \gamma_j) \qquad (b1)$$

$$\sin(\phi_j')y + \cos(\phi_j')x - \|a_i'\| \cos(\theta - \phi_j' + \eta_i') - \|b_j'\| \cos(\phi_j' - \gamma_j') \qquad (b2)$$

Refer to figure (4.1). Here the a_i's are vertices of the "negated" moving polygon ($\bigcirc A$ in Lozano-Pérez [1981, 1983]), in its local coordinate system. η_i is the angle the line from the origin of that coordinate system to the point a_i makes with the coordinate system's x axis, and λ_i is the angle made by the normal to the segment from a_i to a_{i+1}. Similarly the b_j's are the vertices of a convex obstacle polygon, γ_j the orientation of the line from the origin to b_j, and ϕ_j the orientation of the normal to the segment from b_j to b_{j+1}. The parameter θ, a parameter of the configuration space, measures the angle between the x-axes of the object and obstacle coordinate systems.

Type A constraints can be thought of as being generated by a face (edge) of the moving object A coming into contact with a vertex of an obstacle B, and a type B constraint as a vertex of A coming into contact with a face (edge) of B.

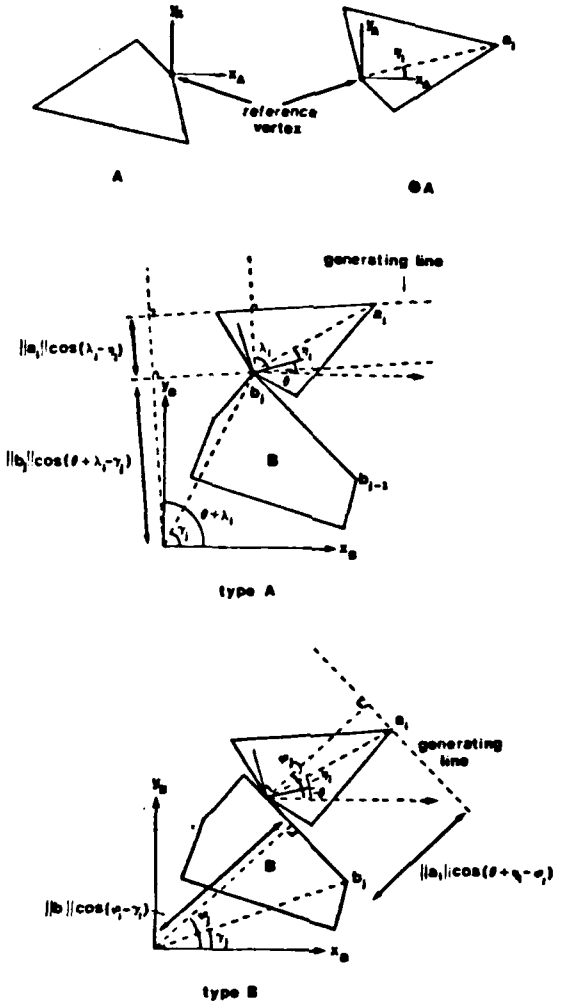


Figure 8. The two types of surfaces can be defined by bring the reference point of the negative of moving object A into contact with a vertex and an edge of fixed obstacle B. Both are defined over a range of orientations θ.

Figure 4.1. An illustration of the terms in equations (a1) and (b1). Reprinted with permission from Brooks and Lozano-Peréz (1983).

Each constraint is valid only over a fixed range of θ. For type A surfaces the range is given by $\theta \in [\phi_{j-1} - \lambda_i, \phi_j - \lambda_i]$ and for type B surfaces by $\theta \in [\phi_j - \lambda_i, \phi_j - \lambda_{i-1}]$.[4]

By applying trigonometric reductions we can express these constraints as follows (only (a1) and (b1) are shown):

$$\begin{aligned}\cos(\lambda_i)Sy + C\sin(\lambda_i)y - \sin(\lambda_i)Sx + C\cos(\lambda_i)x \\ + \sin(\lambda_i - \gamma_j)\|b_j\|S - \|a_i\|\cos(\lambda_i - \eta_i) \\ - C\cos(\lambda_i - \gamma_j)\|b_j\|\end{aligned} \tag{a1}$$

$$\begin{aligned}\sin(\phi_j)y + \cos(\phi_j)x - \|a_i\|\sin(\phi_j - \eta_i)S \\ - C\|a_i\|\cos(\phi_j - \eta_i) - \|b_j\|\cos(\phi_j - \gamma_j)\end{aligned} \tag{b1}$$

Where

$$C = \cos\theta, \quad S = \sin\theta.$$

Now, we can consider a pair of these equations as a system in four variables, (x, y, C, S), and proceed to solve (a1) and (b2), (b1) and (b2), and (a1) and (b1) for x and y. For example, the intersection of two type (a) surfaces, (a1) and (a2) is a curve

$$p : r_{a1} \bigcap r_{a2} \rightarrow \Re^2 \times S^1$$

where $r_{a1} \bigcap r_{a2} \subset S^1$ denotes the intersected applicability constraints for (a1) and (a2). Although the solutions are in the variables C and S, we can use $C = \cos\theta = \cos r$ and $S = \sin\theta = \sin r$ to generate the curve of intersection in $\Re^2 \times S^1$. Because of their excessive length, these equations may be found in appendix I.

4.2.2. Intersecting Trajectories with C-surfaces

A General Discussion for $\Re^2 \times S^1$ and $\Re^3 \times SO(3)$

In order to motivate a discussion of the intersection problem for trajectories and C-surfaces, we now introduce the problem in a context which will be expanded

[4] *Source*: The last three paragraphs are excerpted from Brooks and Lozano-Pèrez, [1983].

upon in chapter 5. The goal of this discussion is to illustrate how the intersection results are used in the planner described in chapter 2.

In principle it is possible to intersect arbitrary trajectories with C-surfaces—such trajectories could translate and rotate simultaneously. Once an intersection is found, we must then determine whether (1) the C-surface is applicable, and (2) whether it lies on the boundary of a *C-Space* obstacle. The question of applicability may be resolved *a priori* by maintaining and updating an accurate set of applicable constraints as the planner moves through rotation space. This set is called the *applicability set*. As the planner moves from Θ to Θ', the updating algorithm must detect which constraints have *expired* (ceased to be applicable) and which new constraints have been *activated* (become applicable). The expired constraints are deleted from the applicability set, and the new constraints are added. In this manner the trajectory will be intersected only with the applicable constraints. Another approach involves intersecting the trajectory with *all* C-surfaces, and then finding the first applicable intersection on the boundary of a *C-Space* obstacle. The first strategy is more general in that it decomposes the image of the trajectory into equivalence classes where the applicability set is invariant. Hence it can in principle be used to map out these equivalence classes on $SO(3)$. However, for most environments the latter strategy runs faster, although both techniques can be shown to have the same asymptotic complexity. Both algorithms have been implemented[5] and tested, and are presented later in chapter 5.

There are also two ways to determine if an intersection lies on the boundary of a *C-Space* obstacle. Let X be the intersection point of a trajectory with an applicable C-surface f. Then X lies on the boundary of a *C-Space* obstacle bounded by f if either of the following holds:

(i) All applicable C-functions in f's family are negative or zero-valued at X.[6]

(ii) If the projection of X into real-space lies within the displaced face of the Minkowski solid corresponding to the generators for f.

Correctness Argument: Let us briefly discuss why (i) and (ii) are equivalent. The correctness of (i) is obvious, since the *C-Space* obstacle is constructed as the finite intersection of half hyperspaces, each of which is defined by a real-valued function on *C-Space*. Let S denote the face of the Minkowski solid, and x the projection of

[5] For $\Re^3 \times SO(3)$ but not for $\Re^2 \times S^1$.

[6] The *family* of a C-function is defined in 3.2.

the intersection point into real-space (i.e., $X = (x, \Theta)$). We will demonstrate that (i) $\leftrightarrow$ (ii).

($\Rightarrow$) Suppose (i), but not (ii). We demonstrate a contradiction. x must lie on the plane of S, even though $x \notin S$, since that is how the C-functions are defined (X could not be an intersection point, otherwise). Recall that the normals of the faces (and planes) bounding the Minkowski solid are defined to be outward-directed from the interior. Since the Minkowski solid is convex, the plane of S bounds a half-space entirely containing the solid. If x is not within S, then it must be outside the plane of some other face, S', which shares an edge with S. But in this case, the C-function corresponding to S' will be positive-valued: a contradiction.

($\Leftarrow$) The Minkowski solid is convex. If $x \in S$, then it is behind (or on) the plane of every other faces of the solid. The C-functions are defined in terms of the distance of x from these planes, which must be negative (or zero). ∎

One further note: suppose that all intersections with C-surfaces--including non-applicable C-surfaces have been sorted along the image of the trajectory in *C-Space*. Then if X is the *first* intersection for which (ii) holds, then f is applicable *and* X lies on the boundary of the *C-Space* obstacle. Again, both approaches have been implemented, and the results are discussed later.

Intersecting Trajectories with C-surfaces in $\Re^2 \times S^1$

We will now present methods for intersecting pure translational and pure rotational trajectories with C-surfaces in $\Re^2 \times S^1$. Note that as long as every path of interest lies entirely within open sets of $\Re^2 \times S^1$, then for every such path there exists a homotopically equivalent path composed of "staggered" pure translations and pure rotations. We assume such paths can be expressed as (piecewise) linear functions of some parameter. Intersecting such a path with a C-surface entails finding the zeroes of the associated C-function (with respect to the parameter).

Pure Translational Paths. Note that (a1) and (b1) are linear in x and y. At a fixed orientation their projection into real space is a line. A pure translational path is also a line. Clearly then, intersection of a pure translational path with a C-surface is trivial.

Pure Rotational Paths. A pure rotational path is a linear function from I^1 to S^1. Intersecting such a path with a C-surface involves finding the zeros (with respect to θ) of the C-function at a constant translation. Observe that C-surfaces (a1) and (b1) are linear in C and S, that is, they can be expressed as

$$E_1 C + E_2 S + E_3 = 0 \tag{4.1}$$

where the terms E_i (for $i = 1, 2, 3$) vary only with x and y. The zeros of (4.1) are not hard to find. First we note that (4.1) can be expressed as a pure quadratic in C (or S), and that solving a quadratic for its zeros is easy. (We must, of course, check for the first applicable zero which is on the boundary of a *C-Space* obstacle). This method is not the best because of susceptibility to numerical problems and singularities. Happily, such equations arise frequently in robot kinematics; Paul (1981) describes a stable, singularity-free calculation for the zeros of exactly this form of trigonometric equation.

Practical Note

The reader will notice that motion sliding along an *intersection manifold* in $\Re^2 \times S^1$ will not in general be a pure translation or rotation. We have not derived the results for intersecting arbitrary trajectories with C-surfaces in $\Re^2 \times S^1$, although in principle it is possible to do so. Note that any such sliding motion can be approximated as closely as desired by a sequence of pure translations and rotations, and furthermore, any such "approximating" planner will be complete (in the sense discussed above) if the "sliding" planner is complete.

Furthermore, our purpose here is a theoretical analysis in low dimensions which still illuminates some of the staggering difficulties in $\Re^3 \times SO(3)$. As it turns out, with the additional degrees of freedom in $\Re^3 \times SO(3)$, this turns out to be considerably less of a restriction.

4.3. Related Problems in $\Re^2 \times S^1$

There are a number of interesting related problems in $\Re^2 \times S^1$. The first addresses techniques for "sliding" along one geometric constraint (C-surface). Sliding is a useful way to circumnavigate obstacles: it can also be used to slide to an

intersection manifold. The second result is of use in the find-space and coordinated motion problems, and involves characterizing the minimum clearance to a C-surface in $\Re^2 \times S^1$. Again, these results are presented not only for their intrinsic interest, but also as an exposition of some of the algebraic techniques required and as an illustration of the complications arising in high-dimensional configuration spaces.

4.3.1. Techniques for Moving Along C-Surfaces in $\Re^2 \times S^1$

In this section we present techniques for moving along a C-Surface. We could imagine using these methods to move to the nearest "edge" (C-Surface intersection), for example. A *level C-Surface* is defined *via* a function $f(x, y, \theta) = k$ for k constant. f is exactly of form $(a1)$ or $(b1)$ (above), and the level surface in $\Re^2 \times S^1$ is all points

$$L = \{ X \in \Re^2 \times r_f \mid f(X) = k \},$$

where $r_f \subset S^1$ is the θ applicability range for f.

Define a hyperplane in $\Re^2 \times S^1$ as the set

$$P = \{ X \in \Re^2 \times S^1 \mid X \cdot H = -h_4 \},$$

where $H = (h_1, h_2, h_3)$.

We intersect the level surface L with the hyperplane P to obtain an intersection curve $p : I^1 \rightarrow \Re^2 \times S^1$. The equation for this curve for both type (a) and (b) C-surfaces may be found in appendix I.

4.3.2. Characterizing Clearance to a C-Surface

It would be very useful to characterize the minimum clearance to a C-surface. The result could be applied in the coordinated motion problem to determine where two mobile objects could possibly interact. In the find-space problem, we could use clearance information to maximize the clearance to a constraint while placing one object, in order to leave room for another. We would like to answer the question:

- *For a point $b_{xy} \in \Re^2$, at what orientation is b_{xy} closest to a C-surface, and what is minimum directed clearance vector at that orientation?*

Using Lagrange multipliers, we can minimize a function $f(x, y, \theta)$ subject to a constraint $g(x, y, \theta) = 0$ by constructing the auxiliary function

$$H(x, y, \theta, \ell) = f(x, y, \theta) - \ell g(x, y, \theta)$$

and simultaneously solving the partial derivatives of H. In our case, g will define a C-surface, and f will be a distance function. Now, the rotational dimensions cannot be treated uniformly in establishing a metric, so we will define distance in Euclidean space. Minimizing the square of the translational distance suffices for our purposes. Hence,

$$f(x, y, \theta) = (x - b_x)^2 + (y - b_y)^2.$$

Differentiating H gives us a system of four equations. Solving these equations for x, y, θ, and ℓ is not trivial. We provide the solutions and their derivation in appendix I. (Solutions are given for both type (a) and type (b) C-surfaces).

4.4. The Intersection Problem in $\Re^3 \times SO(3)$

In this section we extend the previous examples of intersection problems to the 6-dimensional C-space $\Re^3 \times SO(3)$. At this point we must commit ourselves to a particular representation for rotations. The implemented planner uses a rotation matrix specified by Euler Angles. Implementing a different representation for rotations (such as spherical angles, quaternions, or joint angles for a Cartesian Manipulator) would merely require replacing the `Macsyma` rotation abstraction `ROTATE-VECTOR` with the appropriate new function (and recompiling the algebra system). The Euler Angles are

$$\Theta = (\psi, \theta, \phi).$$

The intersection problems in $\Re^3 \times SO(3)$ are as follows. With each problem we give the motivation for attacking it.

(i) Intersecting (level) C-surfaces. (Necessary to construct the intersection manifold).

(ii) Intersecting Level ACFs. (Interesting theoretical question: relates to planning on different kinds of intersection manifolds, and exploiting coherence in *C-Space* constraints).

(iii) Intersecting C-Surfaces with Level ACFs. (Same as (ii)).

(iv) Intersecting Trajectories with C-surfaces. (Indicates that we may have hit a *C-Space* obstacle).

(v) Intersecting Trajectories with ACFs. (Indicates that a constraint has expired (ceased to be applicable)).

Note that we never have to intersect a trajectory with a DAC, since any path straying out of a type (c) constraint's applicability region must first violate an ACF boundary (see Theorem III.5). Since all ACFs can be composed out of type (a) and (b) ACFs, we need only deal with three distinct kinds of functions on $\Re^3 \times SO(3)$ and two on $SO(3)$. In the context of this section the term ACF is used to refer only to the basic type (a) and (b) ACFs out of which all ACFs and DACs may be composed.

Our approach is as follows: We express all C-functions and ACFs in certain canonical forms. The `Macsyma` procedures to derive these forms are provided in an appendix. We then develop certain operations which are defined on any function expressed in these forms. Throughout this discussion of $\Re^3 \times SO(3)$, we use the notation $C_\alpha = \cos\alpha$ and $S_\alpha = \sin\alpha$ where $\alpha \in \{\psi, \theta, \phi\}$. Most of the claims in this section should be self-evident when the rotation matrix $\mathcal{R}(\Theta)$ for Euler Angles is considered.

Claim 4.1: All C-functions are affine in x, y, and z. This is obvious, since $\mathcal{R}(\Theta)$ is a linear transformation. ∎

Claim 4.2: While expressions for C-functions and ACFs can contain cross-terms of the form $C_\alpha S_\beta$, $S_\alpha S_\beta$, or $C_\alpha C_\beta$, it should be clear that $\alpha \neq \beta$, that is, C_α can always be expressed as an affine function of S_α.

To derive this, consider the definition of a C-function (equation (4) in chapter 3) once more:

$$f_p(x, \Theta) = \langle N(\Theta), x \rangle - \langle N(\Theta), (a_i(\Theta) + b_j) \rangle$$

Only the term $\langle N(\Theta), a_i(\Theta) \rangle$ could result in any troublesome terms. For a type (b) constraint, $N(\Theta)$ is a fixed vector. For a type (a) constraint, $N(\Theta)$ is a rotated normal of a face of A, and we have

$$\langle N(\Theta), a_i(\Theta) \rangle = \langle N, a_i \rangle.$$

Finally, for type (c) constraints, $N(\Theta)$ is the cross product of $e_a(\Theta)$ and e_b. This results only in cross-terms of different angles:

$$\begin{aligned}\langle a_i(\Theta), e_a(\Theta) \times e_b \rangle &= \langle a_i(\Theta), \Big(a_{i+1}(\Theta) - a_i(\Theta) \Big) \times e_b \rangle \\ &= \langle a_i(\Theta), a_{i+1}(\Theta) \times e_b - a_i(\Theta) \times e_b \rangle \\ &= \langle a_i(\Theta), a_{i+1}(\Theta) \times e_b \rangle \\ &= \langle e_b, a_i(\Theta) \times a_{i+1}(\Theta) \rangle.\end{aligned}$$

A proof for the ACFs is very similar. ∎

4.4.1. Canonical Forms for C-functions and ACFs

Definition: The *Linear Form* for a C-function $f : \Re^3 \times SO(3) \rightarrow \Re$ is an equivalent expression

$$f(x, y, z, \Theta) = E_1 x + E_2 y + E_3 z + E_4,$$

where $E_i : SO(3) \rightarrow \Re$ (for $i = 1, 2, 3, 4$).

Definition: A *Trigonometric Quadratic Form (TQF)* (in ϕ) for a C-function f is an equivalent expression

$$f(x, y, z, \psi, \theta, \phi) = F_1 \sin\phi + F_2 \cos\phi + F_3,$$

where

$$F_i : \Re^3 \times (\psi, \theta) \rightarrow \Re. \qquad (i = 1, 2, 3)$$

Definition: *A Trigonometric Quadratic Form (TQF)* (in ϕ) for an ACF $g : SO(3) \rightarrow \Re$ is an equivalent expression

$$g(\psi, \theta, \phi) = G_1 \sin\phi + G_2 \cos\phi + G_3,$$

where

$$G_i : (\psi, \theta) \rightarrow \Re. \qquad (i = 1, 2, 3)$$

The TQFs are defined here in ϕ -of course we must also define the TQFs in ψ and in θ in the natural way. ϕ will be our typical example angle in this discussion, however.

Before we proceed let us provide some intuition for these definitions. Imagine deriving a linear form for a C-function, and setting the expression equal to zero. The result is just an expression whose coefficients make explicit how the plane equation of the face of the Minkowski solid changes with rotation.

A TQF (in ϕ) is just a way of expressing C-functions and ACFs in terms of the coefficients of $\sin\phi$ and $\cos\phi$. Linear forms and TQFs will be useful canonical forms for the intersection problem in $\Re^3 \times SO(3)$. It is important to realize that the coefficients E_i, F_i, and G_i are actually functions on the other degrees of freedom.

We see immediately from claims (4.1) and (4.2) that:

Claim 4.3: Every C-function can be expressed as a linear form and as a TQF in ψ, θ, and ϕ; similarly, every ACF can be expressed as a TQF in ψ, θ, and ϕ.

4.1.2. Intersecting C-surfaces in $\Re^3 \times SO(3)$

When intersecting C-surfaces in $\Re^2 \times S^1$, we essentially eliminated variables in a system of equations. This corresponds exactly to "spending" degrees of freedom to comply to two constraints. In $\Re^2 \times S^1$, there were few choices for which variables to eliminate. However, in $\Re^3 \times SO(3)$, we have many more degrees of freedom, and hence there are more choices for how to solve the intersection of a set of constraints. For example, to construct the intersection manifold of three constraints, we could spend all the translational degrees of freedom, which would result in parameterizing the intersection manifold by (ψ, θ, ϕ). Alternatively, we could in principle eliminate the rotational degrees of freedom and parameterize the intersection manifold by (x, y, z). In the former case, we leave (ψ, θ, ϕ) as independent degrees of freedom: parameterizing the intersection manifold simply involves solving the 3 constraints simultaneously for x, y, and z in terms of (ψ, θ, ϕ). To move along their intersection, we are free to plan any values for (ψ, θ, ϕ), and the parameters for the translational degrees of freedom will vary so as to comply to the simultaneous set of constraints. Obviously the choice of which degrees of freedom should parameterize an intersection manifold is important; linear forms and TQFs give us a general way of attacking it. This approach is best illustrated through the following examples:

Example (i). A C-surface in linear form is an expression for a C-function in linear form set equal to zero. Two C-surfaces expressed in linear form may be intersected to yield a 4-dimensional intersection manifold parameterized[7] by (z, ψ, θ, ϕ). This amounts to simultaneously solving the equations

$$f(x, y, z, \Theta) = E_1 x + E_2 y + E_3 z + E_4 = 0$$
$$g(x, y, z, \Theta) = E_1' x + E_2' y + E_3' z + E_4' = 0$$

by first eliminating x and then solving for y. This yields expressions for x and y in terms of (z, ψ, θ, ϕ); we say that (z, ψ, θ, ϕ) form a 4-parameter family for the intersection manifold, and that x and y comply to the C-surfaces f and g as (z, ψ, θ, ϕ) are varied.

This intersection has the following geometric interpretation. Imagine holding orientation constant at Θ_1. Then E_i and E_i' are all constant also. Intersecting f and g at a constant orientation is equivalent to intersecting two planes in $\Re^3$. The intersection is a line, and the position along the line may be parameterized by the one remaining translational degree of freedom, z. The planes intersected are exactly the planes of the faces of the Minkowski solid for f and g at orientation Θ_1.

Example (ii). The intersection manifold $f(X) = g(X) = 0$ from example (i) may be intersected with another C-surface, $h(X) = 0$, expressed in linear form. Suppose z is eliminated. Then the intersection manifold for f, g, and h is parameterized by (ψ, θ, ϕ). The translational degrees of freedom x, y, and z, will be expressed in terms of the rotational degrees of freedom, and will slide along the intersection manifold as rotations are chosen. The new intersection manifold $f(X) = g(X) = h(X) = 0$ is a 3-dimensional sub-manifold of $\Re^3 \times SO(3)$. This intersection has the following geometric significance. Imagine holding orientation fixed at Θ_1 once more. The intersection at a fixed orientation of f, g, and h is the intersection of three planes in $\Re^3$. This intersection (if it exists) is a typically a point. If Θ_1 is allowed to vary, the intersection point moves. The coordinates of the intersection point are the x, y, and z degrees of freedom as they comply to the intersection manifold.

[7] Assume that the constraints are not parallel, and that this is possible, etc.

Examples (i) and (ii) show how to spend translational degrees of freedom to intersect C-surfaces. In (i), we saw that it is possible to plan motion along the the 4-dimensional intersection manifold with one translational and three rotational degrees of freedom. (i) can be used to plan a pure translational path complying to two C-surfaces. The free translational parameter may essentially be chosen to maximize progress in a search algorithm. This is precisely how one "local expert" in the implemented planner works.

One last note on linear forms: the discussion and examples above can be easily generalized to arbitrary level C-surfaces (instead of C-surfaces with level 0) by increasing or decreasing E_4 (the "constant" term in the linear form) by a constant equal to the level.

Intersecting Two TQFs

Consider a TQF g (in ϕ) for either a C-function or an ACF,[8] and suppose further that the TQF has been set equal to zero so that it is actually a *TQF surface*, $\ker g$, by which we mean a TQF for a C-surface or ACF boundary:

$$F_1 \sin\phi + F_2 \cos\phi + F_3 = 0$$

Such a TQF can be expressed as:

$$(F_1^2 + F_2^2)\cos^2\phi + 2F_2F_3\cos\phi + F_3^2 - F_1^2 = 0. \tag{4.2}$$

The new expression is quadratic in $\cos\phi$. (This explains the name TQF). The procedure for intersecting two quadratics is well known.[9] Such a procedure can be used to intersect two quadratics of form (4.2) (i.e., with $\cos\phi$ treated as the quadratic variable). Thus we can obviously intersect any two TQF surfaces. This means that the procedure for intersecting two quadratics can be applied to TQFs of C-surfaces and of ACF boundaries. This immediately yields an effective procedure

[8] Depending on whether the TQF is a C-function or ACF, the functions F_i will have different domains, but this will not matter for our discussion.

[9] For example, see Winston and Horn (1981), (p. 175).

for constructing the intersection manifold of two C-surfaces, two ACF boundaries, or a C-surface and an ACF boundary while spending only rotational degrees of freedom.

4.4.3. Intersecting Trajectories with C-Surfaces and ACF Boundaries in $\Re^3 \times SO(3)$

In this section we extend the method of (4.2.2) for intersecting arbitrary linear pure translational and pure rotational trajectories with C-surfaces in $\Re^3 \times SO(3)$.

Pure Translational Paths. A pure translational path can not stray out of an applicability region. It is not hard to intersect a linear pure translational path with a C-surface. Such a path can be represented as a line in $\Re^3$. At the fixed orientation of the path, any C-surface can be represented as a plane in $\Re^3$. Hence the problem of intersecting a C-surface with a (linear) pure translational path is simply the problem of intersecting a line with a plane. The linear form of any C-surface directly provides the coefficients of this plane for any (applicable) orientation.

Note that in intersecting a pure translational trajectory from some configuration $X \in F$ with a set of applicable C-surfaces, we need only consider C-functions which are positive-valued at X.

Pure Rotational Paths. We restrict our attention to linear, pure rotations in one rotational direction (i.e., in $\pm\hat{\psi}$, $\pm\hat{\theta}$, $\pm\hat{\phi}$), for example,

$$\phi(t) = k_0 + k_1 t \qquad (t \in I^1)$$

(for some constants k_0 and k_1). To intersect such a path with a C-surface (or ACF boundary), we simply find the zeros of the appropriate TQF. For this example, we would use the TQF (in ϕ) for the C-surface:

$$F_1 \sin\phi + F_2 \cos\phi + F_3 = 0. \qquad (4.3)$$

With motion strictly in $\hat{\phi}$, the functions F_i will be constant, and may be regarded simply as the coefficients of a quadratic form. (4.3) is easily solved for the values of ϕ which are its roots (see section 4.2.2). Now, depending on the solution technique,

(4.3) may yield several roots. The correct root may be chosen as follows: for a C-surface, we choose the first root where the C-surface is applicable. For an ACF boundary, we choose the first root where the associated C-surface is applicable. This last step requires examining the other ACFs for the C-surface.

Completeness and Complexity for Rotational Trajectories

We have seen that a continuous path through rotation space can be approximated as closely as desired by a series of linear motions along the rotational axes. We now show that the number of path segments required grows linearly as the resolution of the approximation becomes finer.

Definition: Let V be a vector space, and P and P' be trajectories in V. We say that P' *approximates* P *at resolution* τ if for all $p' \in P'$, the perpendicular distance of p' to P is less than $\frac{1}{\tau}$.

Proposition 4.4: A linear trajectory in a vector space can be approximated by a number of path segments along the axes, which increases linearly as the resolution becomes finer.

Proof: Suppose $V = \Re^3$, and P is a linear trajectory from u to v. Imagine approximating P by linear motions along the $\hat{x}$, $\hat{y}$, and $\hat{z}$ axes. Segment P into k subpaths. From u, attain each of the $k-1$ subgoals (and v) by 3 linear motions (along $\hat{x}$, $\hat{y}$, and $\hat{z}$) from the previous subgoal. This yields a sequence P' of $3k$ motions which approximates P at resolution τ_k. We can bound $\frac{1}{\tau_k}$ from above as follows:

$$\frac{1}{\tau_k} < \frac{1}{k}\max\Big(|v_x - u_x|, |v_y - u_y|, |v_z - u_z|\Big)$$

To achieve a particular resolution τ, it is easy to choose the smallest k satisfying the reverse inequality. We see immediately that k varies linearly with τ. ∎

Let the angle space Q^3 be the domain of a chart for $SO(3)$, as described in chapter 2. Then the angle space trajectory

$$p(t) = \Theta_1 + t\hat{v}$$

for

$$\hat{v} = a\hat{\psi} + b\hat{\theta} + c\hat{\phi}$$

specifies a well-defined trajectory $\mathcal{R}(p(t))$ in $SO(3)$.

Proposition 4.5: We can approximate p as closely as desired by a sequence $\{q_i\}$ of motions in Q^3 along the $\hat{\psi}$, $\hat{\theta}$, and $\hat{\phi}$ directions. Furthermore, the size of the set $\{q_i\}$ grows only linearly as the resolution τ becomes finer.

Proof: Immediate, from proposition (4.4). ∎

4.5. The Algebra System

The treatment here of the implemented algebra system is mercifully brief. Given the discussion, the details, at least in principle, should be easily imagined by most readers. In computer algebra these problems are well understood, and the system does not make a significant contribution to that field. I would like to note, however, that the algebra system is both massive and at the heart of the planning system. It takes 12 hours for a dedicated VAX to optimize and compile the vector form of the constraints (in `Macsyma`) into the primitive functions of the `Lisp` algebra system. On top of these primitives is built a more abstract system, which (for example) can evaluate constraints, intersect constraints, intersect trajectories with constraints, and find zeros of constraints. The intersection and evaluation system has automatic singularity handling (for division by zero, imaginary roots, alignment, etc). For example, to intersect two C-surfaces (*a là* example (i)), the planning system will typically specify a list of preferences for the translational parameterization of the intersection manifold. The system then attempts to construct an intersection manifold with a high-ranked parameterization, and on encountering singularities will back up and try again.

It should now be clear how the algebra system for the planner is designed. For each kind of constraint (C-function or ACF), the algebra system contains procedures which compute the coefficients of the linear form (for C-surfaces only), and coefficients of the TQFs. Each of these procedures can be thought of as a

function of (1) the constraint, and (2) the parameters not explicit in the form (for example, the rotation parameters for the x coefficient of a linear form). On top of this is built a level of abstraction, so that for example the operation "compute the $\sin\phi$ coefficient of the TQF (in ϕ)" is defined on all constraints. Coefficients of all possible forms are described by a total of 12 coefficients for the linear forms of C-functions, 27 coefficients for the TQFs of the C-functions, and 18 coefficients for the TQFs of the ACFs. (These functions correspond exactly to the functions E_i, F_i, and G_i, above). All of these functions are constructed and optimized by `Macsyma` running under `NIL` (Burke (1983)), and then converted into `Lisp`.

We have also experimented with precompiling functions for all possible intersection manifolds (up to some degree).[10] For intersection manifolds of degree 2 or 3, this is not hard, and in fact we have already illustrated all the necessary mathematics in this chapter. Intersection manifolds of higher degrees may be constructed by solving for the submanifold representing the simultaneous satisfaction of several constraints, for example, three constraints in linear form together with two TQFs such as (4.2). When higher degrees are considered, this becomes quite complicated, especially when we allow different parameterizations of the intersection manifolds. Construction of intersection manifolds of higher degree may be easier when different representations for rotations—such as unit quaternions—are employed. This appears a fruitful direction for future research. In practice, we view it as preferable, wherever possible, to obtain the values of coefficients of a form at a certain configuration, and then to plan locally while keeping these coefficients fixed. Thus for example, we might compute the coefficients of the linear forms of two C-surfaces at a given orientation, and then intersect the resulting planes to obtain a translational path along their intersection. The structure of the forms makes this easy to do. For example, rotating the moving object (say, in $\hat{\phi}$) until it hits a constraint is mathematically a complicated operation. All we need do, however, is find the coefficients of the TQF in ϕ, and supply them to a procedure in the algebra system which finds the zeros of TQF surfaces. (But see chapter 5 for the details of the applicability set computation).

[10] The *degree* of an intersection manifold is simply the number of constraints intersected there.

4.6. Related Issues in $\Re^3 \times SO(3)$

4.6.1. Normals to C-surfaces

Let f be an applicable C-function and X a configuration on a level C-surface for f. When an appropriate inner product is defined on the tangent space,[11] the normal to the C-surface at X is the gradient of the C-function f evaluated at f. Normals to C-surfaces are of great importance for motion planning. The gradient may be computed as follows: first the coefficients for the linear form of f (evaluated at X) are obtained:

$$f(X) = E_1 x + E_2 y + E_3 z + E_4.$$

Clearly, $\frac{\partial f}{\partial x} = E_1$, $\frac{\partial f}{\partial y} = E_2$, and $\frac{\partial f}{\partial z} = E_3$. To obtain the partial derivatives in the rotational direction, we find then coefficients of the TQFs (evaluated at X)

$$f(X) = F_1 \sin\phi + F_2 \cos\phi + F_3$$

to obtain

$$\frac{\partial f}{\partial \phi} = F_1 \cos\phi - F_2 \sin\phi.$$

4.6.2. C-functions, Potential Fields, Penalty Functions, and Morse Theory: A Conjecture

A popular approximate algorithm for collision avoidance places "potential fields" around the obstacles (either in real space or in some *C-Space*), and attempts to navigate the reference point through a trough of least resistance. The obstacles may be thought of as having a "charge" which repels the robot, and the goal has an inverse charge which "attracts" it. The potential field method is closely related to the so-called "Morse Theoretic"[12] approach to motion planning, and lends itself to fast control-loop algorithms which can exercise real-time dynamic control of robot arms with few degrees of freedom, in simple environments. As might be expected, the method works best for robots that can be approximated by points or spheres. A proper potential function increases as the robot approaches the obstacle, and goes

[11] See sec. 2.4.2 and Erdmann (1984).

[12] Which takes its name from Morse Theory in differential topology.

to infinity at the obstacle boundary. Traditionally, the potential function is chosen somewhat arbitrarily, with much emphasis on the closeness of the "fit" of the potential surfaces about the real-space obstacles, and with understandable concern for the computability of such functions by specific control hardware. With the theoretical tools we have developed, it is now possible to give a potential function in configuration space which is "exact." For a configuration X, let f be a C-function representing the maximum, applicable, non-redundant constraint from one family. For each such f, we conjecture that a good potential field function would be:

$$P(X) = \begin{cases} (f(X))^{-k}, & \text{if } f(X) > 0, \\ \infty, & \text{if } f(X) = 0. \end{cases}$$

for some $k \geq 2$. Whether or not such penalty functions could be used in devising a fast real-time control algorithm is, of course, another question. The suggestion is primarily intended to show that there is a representation on which (in principle) less approximate potential field methods might be based.

5
Moving Through Rotation Space

5.1. Introduction

In this chapter we discuss some of the computational issues involved in planning paths involving three dimensional rotations. The primary issue is that of keeping track of which constraints (C-functions) are applicable as orientation changes. In principle it is possible to intersect paths with all ACF boundaries, and thus to determine which applicability regions the path traverses and crosses. It is also possible, in principle, to compute the applicability regions *a priori*, before the planning begins.[1] In practice this is computationally infeasible. Even for simple environments, there are typically thousands of constraints, each of which has at least 3 associated type (a) and (b) ACFs. We will investigate alternative strategies which exploit coherence in how the set of applicable functions changes as the robot moves continuously through rotation space. In previous chapters (particularly chapter 4) we showed how to intersect trajectories with C-surfaces and ACFs.

The *applicability set* for an orientation Θ is the set of all applicable constraints (C-functions) there. Clearly, there are regions on $SO(3)$ for which the applicability set is invariant; orientations in the interior of these regions correspond to orientations where no edges or faces of the robot are aligned with the edges or faces of any

[1] This approach is similar to the critical region computations suggested by Schwartz and Sharir (1981).

obstacle. For a fixed orientation Θ, we compute the applicability set by examining the signs of the ACFs for all C-functions (see chapter 3). However, this is clearly not an operation we wish to repeat very often, and the applicability set calculation procedure should be memoized. (A *memoized* procedure records the answer for a given input, so it will not have to be recomputed. Instead, it can simply be looked up in a table). As the robot moves in rotation space, certain constraints will *expire* as the path moves out of their applicability region, and other constraints will *become active* as we move into their applicability region. This suggests that an incremental update algorithm should be possible: we imagine detecting when constraints expire, and when new constraints become active and constructing a *Deletelist* of expired constraints and an *Addlist* of new constraints. The applicability set is then updated by means of the Deletelist and Addlist.

5.2. The Applicability Decomposition for $SO(3)$

In this chapter, we will first present a *naive* algorithm which does not use an update strategy. We then present a more sophisticated procedure, called the $\mathcal{G}$ *update algorithm*, which is an incremental update strategy. We have performed experiments using both algorithms to implement the local operator *Rotate*, which was discussed in chapter 2. Both algorithms have the same asymptotic complexity. Although we have applied both to the find-path problem, they are designed for fundamentally different tasks. The naive algorithm is specialized for a particular find-path operator, while the update algorithm is a general tool for computing a decomposition of *C-Space* for spatial planning.

The $\mathcal{G}$ update strategy addresses the fundamental problem of applicability set computations in a continuous space. Without the $\mathcal{G}$ algorithm, there exists only the "discrete" applicability set computation, which given one point in $SO(3)$ can determine the set of all applicable C-functions. With an incremental update strategy we can map out regions on $SO(3)$ for which the applicability set is invariant. The boundaries of these regions are ACF boundaries. Let $\mathcal{Y}(\Theta)$ be the applicability set at $\Theta \in SO(3)$, and $\simeq$ be a binary relation on $SO(3)$ such that $\Theta \simeq \Theta'$ if, and only if $\mathcal{Y}(\Theta) = \mathcal{Y}(\Theta')$. Clearly, $\simeq$ is an equivalence relation on $SO(3)$, and $SO(3)$ is decomposed by $\simeq$ into disjoint equivalence classes where the applicability

set is invariant. We call this the *applicability decomposition* for $SO(3)$. Computing this decomposition is a fundamental step in reducing continuous spatial planning problems to discrete computational problems.

We will show how to compute these decompositions for sections of $SO(3)$ in any of the directions $S = \{ \pm\hat{\psi}, \pm\hat{\theta}, \pm\hat{\phi} \}$. In particular: The incremental update strategy computes a projection of the applicability decomposition onto a subspace of $SO(3)$ which is isomorphic to S^1. In principle it is not hard to generalize these sections to arbitrary rotational slices: algebraically this entails solving the intersection of a TQF with an arbitrary pure rotation. As we have noticed, any rotation of interest can be approximated as closely as desired be a sequence of rotations in S, with no loss of completeness (at a given resolution).

The naive algorithm, on the other hand, is highly specialized to the particular problem of rotating to a constraint. It does not address the more fundamental problem of decomposing $SO(3)$ into applicability set equivalence regions. We believe that the applicability decomposition is also important to planning problems other than find-path, particularly, for find-space, fine-motion, and planning with uncertainty. However, in practice the naive algorithm has proved faster for rotating to a constraint than any incremental algorithm we have devised. It is gratifying to find that both strategies have the same asymptotic complexity; however, we have no strong indication that the $O(N \log N)$ bound we demonstrate is optimal, and faster algorithms may exist.

5.3. A Naive Algorithm Without an Update Strategy

We begin by presenting a naive algorithm for moving in rotation space which does not employ an update strategy. We wish to design an effective procedure which is to be given a start configuration s, a goal direction $\hat{\alpha} \in \{ +\hat{\psi}, -\hat{\psi}, +\hat{\theta}, -\hat{\theta}, +\hat{\phi}, -\hat{\phi} \}$, and a goal configuration g. The goal configuration differs from s only in that the goal angle in the $\hat{\alpha}$ direction will be g_α instead of s_α. The procedure determines if the robot can reach g along the trajectory in $\hat{\alpha}$, or whether it will strike an obstacle, in which case it must return the C-surface hit and the intersection angle. (We use *intersection angle* to mean the value of α at the intersection point).

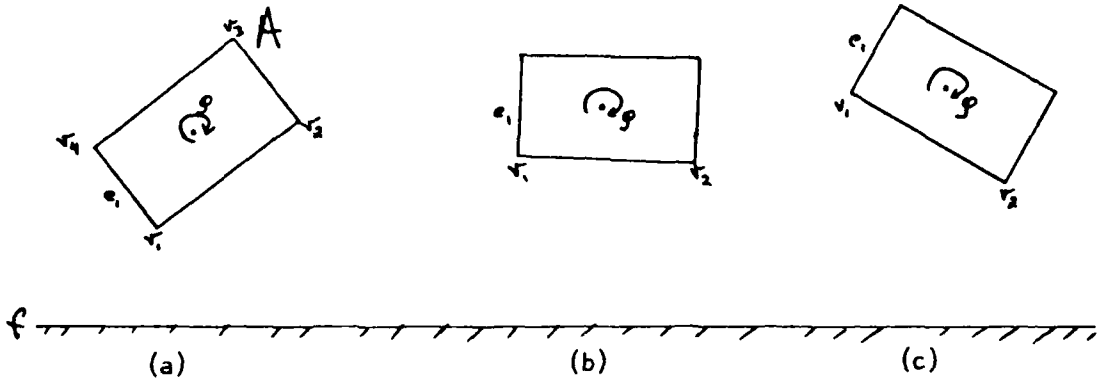


Figure 5.1.

Let C be the set of all C-surfaces. Calculate the intersection of the trajectory

$$p(t) = s + t\hat{\alpha}$$

with every C-surface in C (whether applicable or not). Each such intersection can be expressed as a single angular value (i.e., the value of t or α for which $p(t)$ lies on the C-surface) and hence as a single point on the unit circle. We can order the intersection points by their intersection angle with a C-surface. Sort the intersections around the circle. Then traverse the intersections on the circle in direction $\hat{\alpha}$ from s, and find the first intersection which is both applicable and on the boundary of a *C-Space* obstacle. In 4.2.2 we gave an algorithm for how this may be determined.

5.4. Update Strategies: Example

We now proceed to describe how an update strategy works. If constraints could

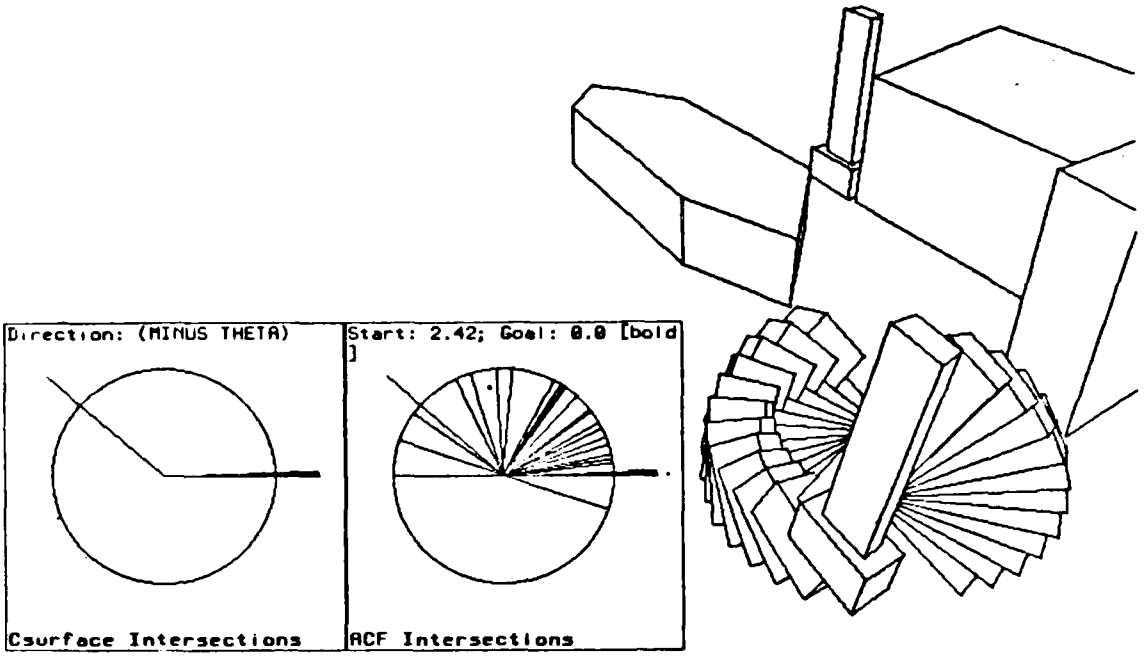


Figure 5.2. As the hammer rotates in the $-\hat{\theta}$ direction from $(\psi_0, 2.42, \phi_0)$ to $(\psi_0, 0, \phi_0)$, the boxes in the lower left show the *C-Space* obstacle boundaries and ACF boundaries that the trajectory hits. Since the hammer is in free-space, it hits no C-surfaces. However, it crosses many ACF boundaries.

expire and become active "arbitrarily", this problem might still be formidable. However the following observation makes things much easier:

Claim (5.1): When a constraint expires, another "neighboring" constraint becomes active.

For example, consider figure 5.1, which depicts a cross-section of a rectangloid A moving above an obstacle face f. As A translates, it rotates in direction $\hat{\phi}$. In 5.1a, constraint (v_1, f) is applicable, and (v_2, f) is not. At 5.1b, however, we move out of the applicability region for (v_1, f) and (v_2, f) becomes active. 5.1b is on the boundary of the applicability regions, and both constraints are applicable. By 5.1c, however, (v_1, f) has expired. (v_2, f) has replaced (v_1, f) in the applicability set. (v_2, f) clearly seems like a neighboring constraint to (v_1, f), in that v_1 and v_2 are adjacent vertices on the edge graph of A. We would like to devise an update

strategy which, given a Deletelist of expiring constraints, could enumerate a small list of candidates for the Addlist. In general an expiring constraint will be replaced by neighboring constraints. However, the neighborhood function is somewhat more complicated than in this simple example. For instance, imagine that A were rotating towards the viewpoint (out of the page), leading with vertex v_3 (see figure 5.4). It is possible for constraints (v_2, f), (v_4, f), and (v_3, f) to replace (v_1, f), if the faces f and $\{ v_1, v_2, v_3, v_4 \}$ are parallel when (v_1, f) expires. Clearly v_3 is also "near" v_1, but not as near as v_2 and v_4. To exploit claim (5.1), it remains to be seen just what we mean by a "neighboring constraint." We should emphasize that the update strategy does not predict exactly which constraints will become active, but merely a set of candidate constraints, some of which must replace the expiring constraints in the applicability set.

5.5. Using Update Strategies

Let us modify the naive algorithm to incorporate an update strategy. At configuration s, we compute the applicability set. The trajectory p is next intersected with all C-surfaces in the applicability set, and with all ACF boundaries for these C-surfaces. The two lists of intersections are merged and sorted around the unit circle. (The sort key, once more, is the intersection angle). We call this sorted structure of C-surface and ACF intersections the *intersection queue*, since it a priority queue containing intersections. An entry in the intersection queue is a pair:

$$\Big(\text{C-surface or ACF}, \text{Angle of intersection}\Big).$$

We then traverse the intersection queue in order from s in direction $\hat{\alpha}$, taking the following actions when we encounter a C-surface or an ACF intersection:

(1) When an ACF boundary is hit, a C-surface has expired. Let the angle of intersection be α_l. Sometimes several C-surfaces expire at once; in this case their ACF boundaries will all have the same intersection angle on the circle. Determine all the C-surfaces that expire at α_l (simply scan down the intersection queue until an intersection angle greater than α_l (with respect to direction $\hat{\alpha}$) is found). These C-surfaces constitute the Deletelist. Assume we have an update procedure, which

can determine an Addlist of newly active C-surfaces given a Deletelist of expiring C-surfaces. Call the update procedure with the Deletelist, to determine the Addlist.

(i) Delete all C-surfaces in the Deletelist from the Applicability set.

(ii) Delete all C-surfaces in the Deletelist from the intersection queue.

(iii) Delete all the ACFs of C-surfaces in the Deletelist from the intersection queue.

(iv) Create an Addlist intersection queue, i.e., a sorted structure containing the intersections of all C-surfaces in the Addlist, and all ACF boundaries of these C-surfaces, with the trajectory p.

(v) Merge the Addlist intersection queue with the old intersection queue.

(2) When encountering a C-surface intersection, we know the C-surface must be applicable, since we have not yet hit an AC[illegible]undary which could invalidate it. (This is essentially the correctness criterion maintained by step (1) of the algorithm). Test to see if the intersection is on the boundary of the *C-Space* obstacle. Note that this operation typically requires knowing the applicability set.

We then continue traversing the intersection queue (of course, resuming traversal the next α slightly beyond α_I in the $\hat{\alpha}$ direction) until either an obstacle *is hit or the goal angle is reached.* As the intersection queue is traversed, steps (1) and (2) are performed to update the queue and detect collisions whenever an ACF-boundary or C-surface (respectively) is crossed. $SO(3)$ is typically quite dense in ACF boundaries: see figure 5.2. In this figure, the small boxes depict one dimensional slices (isomorphic to S^1) of rotation space in the $-\hat{\theta}$ direction. The thin line extending out of the circle indicates the start angle, which is $\theta = 2.42$ radians, and the heavy line extending out of the circle indicates the goal angle, which is $\theta = 0$. The intersections of the trajectory with C-surfaces are shown in the left box (there are none). The intersections of the trajectory with ACF boundaries are shown in the right box. Each line indicates the angle of intersection for an ACF boundary. The applicability set is invariant between intersection points. The moving object is shown rotating between the start and goal angle. The C-surfaces and ACFs were generated by the moving object and obstacles shown. However, the actual size of each Addlist is usually small. The algorithm works by maintaining a correct applicability set as we move in $\hat{\alpha}$, and by modifying the intersection queue to remove C-surface and ACF intersections that are not applicable.

5.6. Update Strategies

In this section we finally discuss specific update strategies. An update strategy has two parts: first, given a Deletelist of expiring constraints, it must predict a set of C-functions guaranteed to contain the Addlist. Second, it must test each of these predictions to determine which are really applicable. The latter operation is conceptually trivial, but since it is expensive, we wish to make the prediction set as small as possible. For example, predicting C, the set of all C-functions is clearly correct, but not very useful.

A better approximation would be as follows: given a Deletelist, determine all the C-families (i.e., families of C-surfaces) it represents. A safe prediction would comprise all the C-functions in these families, since clearly an expiring constraint will be replaced by another constraint from its own family. In practice this approximation has proved useful, however, it is not the best we can do. In particular, note that even two cuboids will generate 48 type (a) C-surfaces, 48 type (b) C-surfaces, and 144 type (c) C-surfaces. Clearly the C-family approximation is not a very tight upper bound for the replacement set, that is, the Addlist for a Deletelist.

Let $\mathcal{V}_P$, $\mathcal{E}_P$, and $\mathcal{F}_P$ denote the vertices, edges, and faces of polyhedron P. For a moving polyhedron A and an obstacle polyhedron B, we can express the family of constraints as:

$$(\mathcal{F}_A \times \mathcal{V}_B) \bigcup (\mathcal{V}_A \times \mathcal{F}_B) \bigcup (\mathcal{E}_A \times \mathcal{E}_B).$$

To be formal, this should, strictly speaking, be considered the domain of a bijection C which maps pairs of generators to the function space of C-functions, but where there is no ambiguity we will speak of a pair (g_A, g_B) as representing the corresponding C-function $C(g_A, g_B)$.

For an expiring C-function (g_A, g_B), we would like to define a neighborhood map on a polyhedron P,

$$\mathcal{G} : \mathcal{V}_P \bigcup \mathcal{E}_P \bigcup \mathcal{F}_P \rightarrow (\mathcal{V}_P \bigcup \mathcal{E}_P \bigcup \mathcal{F}_P)^*$$

(where * is the Kleene star denoting closure) such that the set

$$\mathcal{G}(g_A) \times \mathcal{G}(g_B) \tag{5.1}$$

is the smallest maximal replacement set for (g_A, g_B). In other words, we want (5.1) to contain all possible replacement sets for (g_A, g_B), no matter what the rotational motion; but we also wish (5.1) to be as small as possible so as to minimize the ACF computations. It is possible for $\mathcal{G}$ to be local in character: although several constraints in a family may expire simultaneously, all that we require is that the union of their replacement sets is correct.

We conjecture it might be possible to find exact–or at least smaller replacement sets by taking the specific motion into account. Such a strategy has not yet been developed, however.

5.6.1. Mathematical Preliminaries

In chapter 3, we gave an informal definition (by example) of the boundary and coboundary operators. We now define and employ two related operators which can be composed to define operators such as "the faces which contain vertices v_1, v_2, and v_3" and "the edges which are incident at the vertices of these faces."

In this section we define the *discrete* boundary and coboundary operators. Consider a finite collection of cells, S. The *discrete boundary* and *discrete coboundary* of S, denoted $\hat{\partial}S$ and $\hat{\delta}S$, are defined as follows:

$$\hat{\partial}S = \bigcup_{s \in S} \partial s$$
$$\hat{\delta}S = \bigcup_{s \in S} \delta s.$$

The discrete boundary and coboundary operators have very different properties from the normal boundary and coboundary operators. For example, if f is a face, then $\partial^2 f = 0$, while $\hat{\partial}^2 f = vert(f)$. To see this, observe that

$$\begin{aligned}\hat{\partial}^2 f &= \hat{\partial}(\hat{\partial} f) \\ &= \bigcup_{e \in \hat{\partial} f} \partial e \\ &= vert(f).\end{aligned}$$

In fact, for any "well behaved" object P (and in particular, any polytope), $\partial^2 P = 0$ and $\delta^2 P = 0$ (this is a fundamental theorem of topology). However, two (or more) applications of the discrete boundary or coboundary operator will not, in general, yield 0.

Examples: $\hat{\delta}^2(v_1, v_2, v_3)$ is the set of faces F which contain at least one of the vertices v_1, v_2 or v_3. Since for one face f, $\hat{\partial}^2 f = vert(f)$, then $\hat{\partial}^2 F = \hat{\partial}^2 \hat{\delta}^2(v_1, v_2, v_3)$ is the vertices of all the faces F. The set of edges incident at these vertices is $\hat{\delta}\hat{\partial}^2\hat{\delta}^2(v_1, v_2, v_3)$.

Exercise: What is $\hat{\partial}^2\hat{\delta}^3(v_1, v_2, v_3)$?

Elementary Review: Boundary, Coboundary, and Star

We must show that the discrete boundary and coboundary operators are well behaved. We will do so by presenting a formal definition of $\hat{\partial}$ (and $\hat{\delta}$) on a single chain. Readers who have encountered a bit of homology will find the demonstration transparent. Others may wish to take this section on faith, and to skip to the next section, where we define the star operator.

Discrete boundary and coboundary operators can be considered as the ordinary boundary and coboundary "modulo orientation." We see this as follows. (For a more comprehensive account see any textbook on elementary topology, for example, Hocking and Young (1961)).

Let K be an arbitrary oriented complex of abstract cells, and Z an arbitrary (additively written) abelian group. An n-dimensional chain on the complex K with coefficients in Z is a function c_n mapping oriented n-cells of K to Z, such that if $c_n(+\sigma^n) = z$, then $c_n(-\sigma^n) = -z$. An arbitrary n-chain c_n on K can be written as the formal linear combination

$$\sum_i z_i \cdot \sigma_i^n,$$

where $z_i = c_n(+\sigma_i^n)$. The boundary operator ∂ is a mapping from n-chains to $(n-1)$-chains. $\partial(z_i \cdot \sigma_i^n)$ is an $(n-1)$-chain which has non-zero coefficients only on the $(n-1)$-faces of the cell σ_i^n. Formally, let $[\sigma^n, \sigma^{n-1}]$ be the *incidence number* for σ^n and σ^{n-1}, that is

$$[\sigma^n, \sigma^{n-1}] = \begin{cases} 0, & \text{if } \sigma^{n-1} \text{ is not a face of } \sigma^n, \\ +1, & \text{if } \sigma^{n-1} \text{ is a positively-oriented face of } \sigma^n, \\ -1, & \text{if } \sigma^{n-1} \text{ is a negatively-oriented face of } \sigma^n. \end{cases}$$

Hence,

$$\partial(z_i \cdot \sigma_i^n) = \sum_{\sigma^{n-1}} [\sigma^n, \sigma^{n-1}] \cdot z_i \cdot \sigma^{n-1}.$$

To factor out the effect of orientation, we define the discrete boundary operator as follows:

$$\hat{\partial}(z_i \cdot \sigma_i^n) = \sum_{\sigma^{n-1}} |[\sigma^n, \sigma^{n-1}]| \cdot z_i \cdot \sigma^{n-1}.$$

Discrete coboundary is defined analogously.

The Star Operator

Let P be a polyhedron. Any cell k is a *face* of itself, although it is not a *proper* face. A proper face of P must be lower in dimension than P: If an n-dimensional cell k is on the boundary of P, then we call k an proper n-face of P. Thus edges are proper 1-faces, and vertices proper 0-faces of a 3-dimensional polyhedron. Let K be some complex of cells. If k is a n-face of K, then we write $K > \mathrm{k}$. We will usually assume that a face is a proper face.

Now, let Σ be some set of cells in K. The *star* of Σ (in K) is defined by

$$\mathrm{St}(\Sigma, K) = \{\, \sigma \in K \mid (\exists\, \tau \in \Sigma),\, \sigma > \tau \,\},$$

i.e., the set of all cells in K that contain a member of Σ in their boundary. When there is no ambiguity we will simply write $\mathrm{St}(\Sigma)$. (Giblin (1977), Hocking and Young (1961)).

For a cell $\mathbf{k}$, define $\hat{\delta}^0\mathbf{k} = \mathbf{k}$, $\hat{\delta}^1\mathbf{k} = \hat{\delta}\mathbf{k}$, and $\hat{\delta}^2\mathbf{k} = \hat{\delta}(\hat{\delta}\mathbf{k})$, (etc). We see immediately that the star of $\{\,\mathbf{k}\,\}$ may be computed as

$$\mathrm{St}(\{\,\mathbf{k}\,\}) = \bigcup_{i=0}^{n} \hat{\delta}^i\mathbf{k}.$$

Using this observation, we have implemented the star operator by recording the boundary and coboundary of each cell in the geometric model.

5.6.2. Local Computation of Replacement Sets

Type (a) and (b) Constraints

Consider figure 5.3. (v_1, f) denotes a type (b) constraint. Consider any rotational motion from the configuration shown. Assume this rotation will cause (v_1, f) to expire. We wish to determine the maximal possible type (b) replacement set for (v_1, f), that is, the set of neighboring type (b) constraints which could replace (v_1, f) under any conceivable rotation.

Consider the set

$$\Big(\hat{\partial}(\hat{\delta}v_1) - \{\, v_1 \,\}\Big) \times \{\, f \,\}. \tag{5.2}$$

$\hat{\delta}v_1 = \delta v_1$ is just the edges which meet at v_1. The discrete boundary of these edges is simply the collection of their vertices. v_1 is deleted, since it is expiring. Now, consider a rotational motion which causes (v_1, f) to expire. (5.2) will contain replacement type (b) constraints. However, (5.2) is not maximal: consider a rotation which causes (v_1, f) to expire at some orientation at which a face f' containing v_1 is parallel to f. (See figure 5.4). Then all the vertices of f' are replacements for v_1,

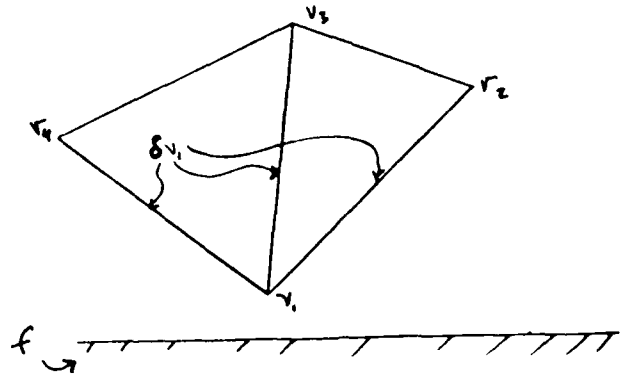


Figure 5.3.

that is, the type (b) replacement set is $\Big(vert(f') - \{ v_1 \}\Big) \times \{ f \}$. In general, the maximal predicted type (b) replacement set for (v_1, f) is

$$\Big(vert(\hat{\delta}^2 v_1) - \{ v_1 \}\Big) \times \{ f \}. \tag{5.3}$$

By similar analysis, we see the following claims:

Claim (5.3): When (v_1, f) expires, so will at least one type (c) constraint with generators in

$$\hat{\delta} v_1 \times \hat{\partial} f.$$

Claim (5.4): See figure 5.5. When a type (c) constraint (e_a, e_b) expires, so must some type (b) constraint with generators in

$$\hat{\partial} e_a \times \hat{\delta} e_b.$$

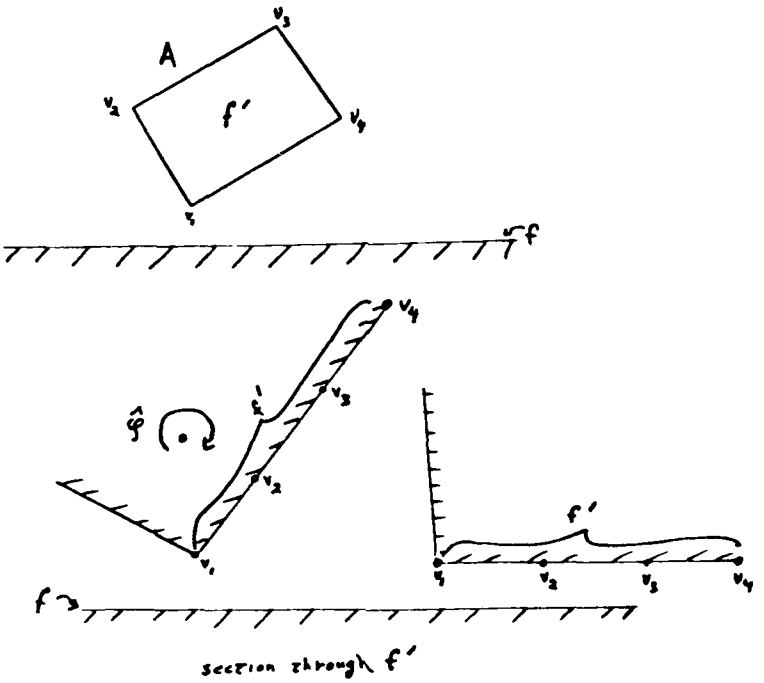


Figure 5.4. (a) A is rotating above face f, out of the page, (towards the eyepoint). f' is the visible face, with vertices v_1, v_2, v_3, and v_4. (b), (c) show a section through f' as A rotates. When f and f' are parallel, v_2, v_3, and v_4 all become active as v_1 expires. This is a singular point; as A continues to rotate, v_2 and v_1 expire, and v_3 remains applicable. The instantaneous replacement set for v_1 is $vert(f') - \{ v_1 \}$.

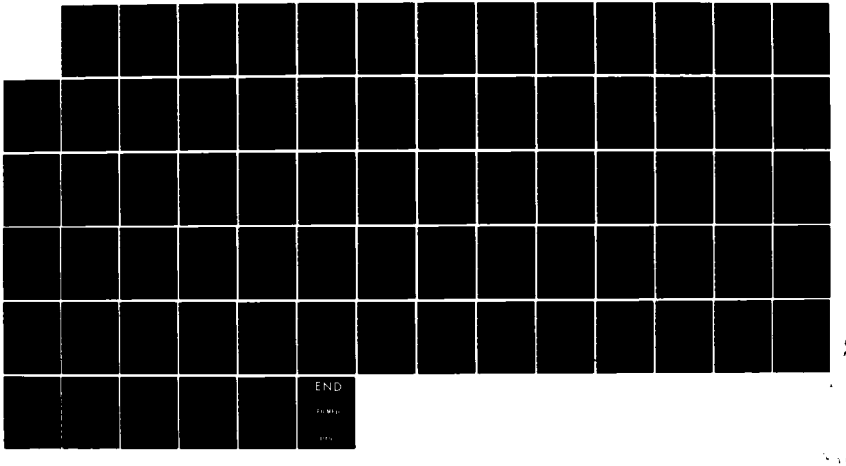
This analysis is, of course, symmetric for type (a) constraints. In this case, f would be interpreted as a face of A and v_1 as an obstacle vertex. The equations given all work when the generator pairs are reversed.

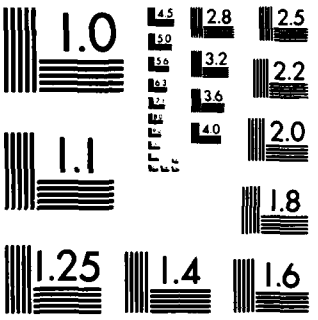
Claims (5.3) and (5.4) are particularly interesting, in that they suggest that we can detect all expiring type (c) constraints by examining the ACFs of type (a) and (b) constraints alone.

5.6.3. Definition of the Neighborhood Mapping for the Replacement Generators

The replacement set in (5.3) makes a certain amount of sense: the replacements for an expiring generator v_1 are to be found in the faces containing v_1. On a polyhedron P, the general neighborhood function $\mathcal{G}$ is a simple generalization of

AD-A150 312 MOTION PLANNING WITH SIX DEGREES OF FREEDOM(U) 3/3
MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL
INTELLIGENCE LAB B R DONALD MAY 84 AI-TR-791
UNCLASSIFIED N00014-81-K-0494 F/G 6/4 NL





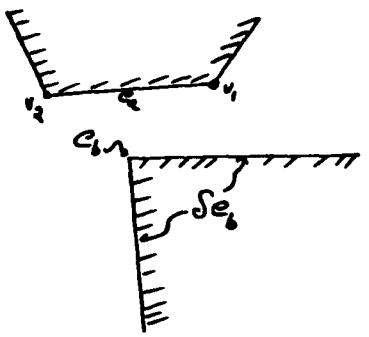


Figure 5.5. Section through e_b. $\hat{\partial} e_a = \{v_1, v_2\}$.

(5.3):

$$\mathcal{G}(\cdot) = \mathrm{St}(vert(\cdot), \partial P),$$

that is, $\mathcal{G}(\mathrm{k})$ is the set of all cells which contain vertices of k as faces.

Let D be a Deletelist. The smallest maximal replacement set for D is

$$\bigcup_{(g_A, g_B) \in D} \mathcal{G}(g_A) \times \mathcal{G}(g_B).$$

This particular formulation requires that we ignore "nonsense" pairings such as all members of $\mathcal{V}_A \times \mathcal{V}_B$. This is easily accomplished by appropriate construction of the function C mapping pairs of generators to the function space of C-functions. We extend the domain of C to $\hat{\partial} A \times \hat{\partial} B$, and map all generator pairs except those in $(\mathcal{F}_a \times \mathcal{V}_B) \cup (\mathcal{V}_A \times \mathcal{F}_B) \cup (\mathcal{E}_A \times \mathcal{E}_B)$ to $\emptyset$.

A Correctness Proof for $\mathcal{G}$

We shall now argue that $\mathcal{G}$ is the correct mapping to predict smallest maximal replacement sets. Let (g_A, g_B) be an expiring constraint at some orientation Θ. The replacement set for a constraint is the Cartesian product of the the replacement sets for its generators. Hence the the replacement set for (g_A, g_B) is $r(g_A) \times r(g_B)$. Consider the replacement set $r(g_A)$ for g_A, i.e., the set of constraints

$$r(g_A) \times \{ g_B \}$$

which will replace (g_A, g_B). This set of constraints will become active at orientation Θ, while (g_A, g_B) expires. Let

$$G = \{ g_A \} \bigcup r(g_A).$$

Note that (1) all constraints in $G \times \{ g_B \}$ must be applicable at orientation Θ, and (2) Θ must lie on the boundary of each of their applicability regions in $SO(3)$. We say that at Θ each $g \in G$ is in ACF boundary condition. Note further that $r(g_A)$ is not a predictive replacement set, but any actual replacement set for a generator g_A under some arbitrary rotation.

We will first show that all $g \in G$ are coplanar.

All ACFs are defined in terms of a *contact vertex* and an *applicability vertex* (see chapter 3). The contact vertex is brought to rest on some *applicability plane* (which is parallel to a face of the other polyhedron), and the applicability vertex is constrained to lie above that plane. When a constraint is in ACF boundary condition, then both the contact vertex and the applicability vertex of at least one of its ACFs are constrained to lie on the applicability plane. (We consider type (c) constraints to be composed of four such ACFs, two of which are type (a) and two or which are type (b) ACFs). In addition, observe that each line segment

$$\Big(\text{Applicability vertex, Contact vertex}\Big)$$

lies on some edge of either polyhedron A or polyhedron B. In fact, these edges, which we term *applicability edges*, cover the edge graphs of both polyhedra, although the mapping is many-one. We are given a generator g_A for a constraint (g_A, g_B). The constraint is placed in ACF boundary condition. This requires aligning an (applicability) edge of A with a face of B (or vice versa). (This point is fundamental to understanding the correctness argument: if both contact vertex and applicability vertex must lie on the applicability plane, then the applicability edge, which is an actual edge of A, must be aligned with the plane). We are then asked to find all constraints which can be simultaneously placed in ACF boundary condition. This is equivalent to asking, "Given one edge of A aligned with some face of B, and maintaining this alignment, what additional edges can simultaneously be aligned with faces of B, such that all associated constraints are in ACF boundary condition?" (By *associated constraints* we mean the following: the aligned edge is considered as an applicability edge. Since the applicability edges cover the edges of the polyhedra, the associated constraints for an applicability edge e_A are those C-functions for which the orientation of e_A determines applicability.)

Now, by fixing an edge e_A at some arbitrary aligned orientation (with a face of B), we retain one rotational degree of freedom $\hat{\varphi}$ about e_A. We wish to choose this rotation such that (1) the constraint associated with e_A (i.e., (g_A, g_B)) remains applicable (and of course, in boundary condition), and (2) a maximal set of constraints is simultaneously placed in boundary condition. The replacement set we compute is the union of these maximal sets. (2) requires a maximal number of additional edge alignments, and must also preserve the disjoint interior criterion. So choosing $\hat{\varphi}$ so as to maximize the number of edge alignments propagates the alignment constraint. Clearly, by propagating the alignment constraint, we obtain a set of coplanar edges (recall that A and B are convex). Each edge represents a contact vertex and an applicability vertex for (one or more) ACFs in applicability boundary condition. The associated generators must also be coplanar.

We have seen that all replacement generators $r(g_A)$ must be coplanar with g_A. (As usual, there exists a symmetric argument for $r(g_B)$). Given an expiring generator g_P on a polyhedron P, we wish to predict replacement sets. Replacement

sets are obtained from maximal sets of coplanar generators which contain vertices of g_P. Clearly, the maximal coplanar sets of generators for a convex polyhedron are exactly its faces (and their boundaries). Hence, to predict replacement sets, we must find the set of faces of P (along with their boundaries) that contain vertices of g_P. This set is

$$\mathcal{G}(g_P) = \mathrm{St}(vert(g_P), \partial P). \quad \blacksquare$$

5.7. Analysis and Evaluation

We have implemented algorithms for moving in some selected rotational direction until either the goal or a C-surface is reached: The *naive algorithm* (see section 5.3), the *predictive update* algorithm based on the C-surface family as a loose maximal bound on the replacement set (section 5.6), and the *incremental update* algorithm based on $\mathcal{G}$ (section 5.6.3). We next show that the naive algorithm and the $\mathcal{G}$ algorithm both have the same asymptotic complexity. This means that their performance will largely depend on the constant factors in the computation. We discuss empirical results to indicate the size of these constants.

5.7.1. Complexity

Naive Algorithm: $O(n \log n)$

Let N be the number of C-surfaces in the environment (including non-applicable C-surfaces). If the moving object is made up of m_0 convex polyhedra with k_0 generators each, and the obstacle environment comprises n_0 convex polyhedra with j_0 generators each, then clearly $N = j_0 k_0 m_0 n_0$. The complexity of the naive algorithm is as follows:

(i) Intersect trajectory with all C-surfaces ($O(N)$).

(ii) Sort intersections around S^1. ($O(N \log N)$).

(iii) For each intersection, determine if it is applicable and on the boundary of a *C-Space* obstacle. First, test to see if the C-surface is applicable by examining its ACFs. If so, there are two options: (1) if the applicability set is known at the intersection point, we can test to see if the other C-functions in the family are negative or zero. (2) If the applicability set is not known at the intersection point, we can compute the displaced face of the Minkowski solid corresponding to the two generators for the C-surface. Next, test to see whether the intersection point falls within the face. (1) would make this step $O((j_0 k_0)^2 m_0 n_0) = O(j_0 k_0 N)$.

However, (2) needs only examine the generators of a constraint, and allows this step to be $O(N)$. $(O(N))$.

We see that the complexity of the naive algorithm is $O(N \log N)$.

$\mathcal{G}$ Update Algorithm: $O(N \log N)$

The complexity of the $\mathcal{G}$ update algorithm is as follows:

(i) Intersect trajectory with all applicable C-surfaces and their ACFs. Let the number of applicable C-surfaces be $\frac{N}{k} < N$, and the number of ACFs per C-surface be a. $(\frac{N}{k}(1+a)$ intersections $= O(\frac{N}{k}))$.

(ii) Sort the intersections around S^1. $(O(\frac{N}{k} \log \frac{N}{k}))$.

(iii) For each intersection: Sort, add, and delete j C-surfaces from the intersection queue. $(O(j \log j))$.

This yields complexity:

$$\frac{N}{k} \log \frac{N}{k} + \sum_{i=1}^{\frac{N}{k}} j \log j = \frac{N}{k}(\log N - \log k) + \frac{N}{k}(j \log j)$$
$$= O(N \log N).$$

In the next section, we justify treating k as a constant.

Actual Performance

In practice, the naive algorithm has run faster than the $\mathcal{G}$ update algorithm for the specific problem of rotating until a C-surface (or the goal) is reached. As the complexity analysis has shown, since both algorithms are $O(N \log N)$, the difference in performance will be due to different constant factors. A good estimate for k is 10. For example, in a typical environment with 624 type (a), 704 type (b), and 1872 type (c) C-surfaces, 4 sample applicability sets have sizes 353, 362, 365, and 355. j is quite small; for this environment it is typically between 2 and 40. The number of ACFs per C-surface depends on the degree of the vertices. For trihedral vertices, for example, $a \leq 4$ (type (c) C-surfaces have 4 ACFs). Hence this tends to balance out any possible gains, since k is not much bigger than a.

Once more we should remember that the $\mathcal{G}$ update algorithm is designed to solve the more general problem of applicability decomposition of $SO(3)$, while the naive algorithm has been specialized to solve the "rotate to a C-surface" problem.

We speculate that similar specialized algorithms may be developed as fast solutions to specific spatial planning problems. However, decomposition tools are a more general solution which can be applied to a whole class of spatial planning problems.

5.7.2. Related Work, Searching and Lazy Evaluation

The implemented planning system is described in chapter 2. The control structure of the algorithm is a search. The search employs certain local operators for moving between configurations. One such local operator is precisely the "rotate to a C-surface (or the goal)" algorithm.

As for most heuristic[2] search algorithms, an adversary can probably devise a find-path problem which must require an exponential amount of time to solve. This does not imply that a polynomial-time algorithm using the mathematics presented in this thesis could not be devised; indeed, the theoretical work of Schwartz and Sharir (1982a) suggests this possibility. However, in practice, the planner has performed quite well. We offer the following explanation for why the planner should, in average cases, perform better than in the adversary situation.

In the theoretical work of Schwartz and Sharir (1982a) and Schwartz and Sharir (1981), the concept of *non-critical regions* is introduced. A non-critical region, intuitively speaking, is a region in (free) configuration space where the constraints are invariant. We employ similar constructs via sets of non-redundant constraints, and by means of applicability sets. In Schwartz and Sharir (1981), for example, free space is decomposed into critical and non-critical regions, and the connectivity of these regions computed. The connectivity graph is then searched for a path. However, computing these regions is (geometrically and algebraically) quite difficult. The regions are at least as complex as the *C-Space* obstacle and applicability regions.

Instead of precomputing the applicability regions (or knowing them *a priori*), our planner computes them as it explores configuration space. While in the worst case the entire applicability decomposition must be calculated, this case does not

[2]We use the term heuristic in reference to the time complexity, and not the completeness of the algorithm.

arise in practice. We effectively adopt a policy of *lazy evaluation* of applicability in devising the planning algorithm.

6
The C-Voronoi Diagram and its Relationship to Intersection Manifolds

6.1. Introduction

For a finite set of points P in the plane, the *Voronoi diagram* is the set of all points in the plane which are equidistant from two or more points in P. The Voronoi diagram for P is a network of straight line segments. Drysdale (1983) introduced the *generalized Voronoi Diagram* (or GVD) for the plane: for a set of polygons in the plane, the GVD is defined to be all points in the plane which lie (perpendicularly) equidistant between two or more polygons. The GVD is a network of straight line segments and parabolic sections. If the polygons are considered as obstacles, the GVD represents the network of paths through free-space which maximize clearance from the obstacles. Brooks (1983a) and Ó'Dúnlaing and Yap (1982), Ó'Dúnlaing, Sharir and Yap (1982) have developed definitions and algorithms employing an extension of the Voronoi diagram for low-dimensional configuration spaces. Nguyen (1983) also discusses the relationship of global methods to the GVD.

More formally, the generalized Voronoi diagram (and its extensions) decompose the free space into a set of regions, $\{ R_i \}$, such that all points $X \in R_i$ are closer to one obstacle than to any other. Thus points on the GVD are equidistant from two or more obstacles.

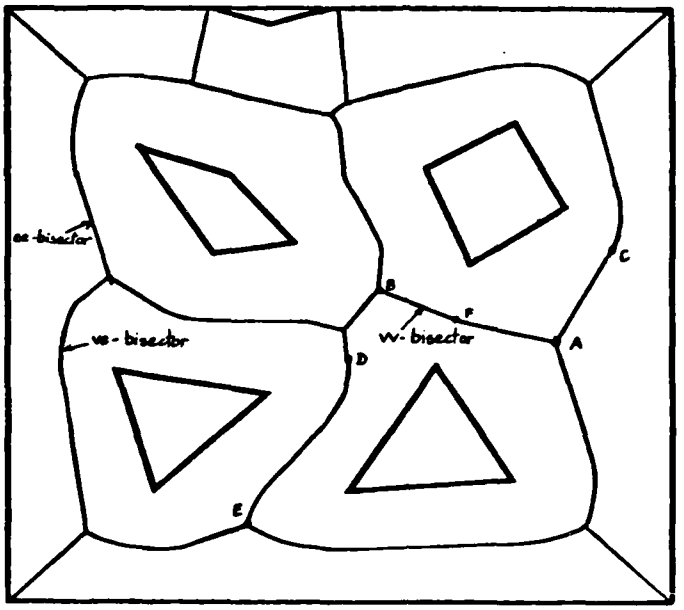


Figure 6.1. A picture of the generalized Voronoi Diagram for a bounded 2D workspace containing four polygonal obstacles. Reprinted with permission from Nguyen (1983).

In this chapter, we extend the concept of the generalized Voronoi diagram to the six dimensional *C-Space* $\Re^3 \times SO(3)$, to provide a formal, constructive definition of the *C-Voronoi Diagram*, or *CVD*. The CVD is an attractive construction, in that it contains a representative component for each "branch" of free space. Each such component is submanifold of dimension $0 \leq d \leq 5$, called a *Voronoi manifold*. We will derive the following connection between intersection manifolds and the CVD:

> Let p be a path along the CVD. p lies along a connected chain of Voronoi manifolds, $\mathcal{V}_1, \ldots, \mathcal{V}_k$. We demonstrate that for each Voronoi manifold $\mathcal{V}_i$, there exists an equivalent intersection manifold of level C-surfaces, I_i. Furthermore, we also show that for every connected chain of Voronoi manifolds, there is an equivalent connected chain of intersection manifolds (of level C-surfaces). (The equivalence we demonstrate is actually stronger that homotopic equivalence).

This yields an immediate (theoretical) completeness result for planning along

intersection manifolds. While our proof is constructive, it cannot be considered an effective procedure. The charts for the Voronoi manifolds are undoubtedly very difficult to derive. This in turn makes it hard to develop planning algorithms along the C-Voronoi diagram. In *C-Space*, the most attractive feature of the CVD is not that it maximizes clearance from obstacles, but that it represents the connectivity of free space. In other words, given the CVD, the Movers' problem can be solved by connecting the start and goal configurations to the same connected component of the CVD. But since the Movers' problem has already been reduced to the task of navigating a point, it is clear that, modulo some uncertainty bound, we do not need to maximize clearances while in planning paths in *C-Space*. We demonstrate that instead, it is possible, in principle, to devise a planning algorithm along intersection manifolds—for which we have derived charts (chapter 4)—which is equivalent to a planner along the CVD.

Generalized Voronoi Manifolds

In this section we define the *C-Voronoi Diagram* (*CVD*) for the configuration space $\Re^3 \times SO(3)$. Note that for Euclidean configuration spaces we would employ the standard techniques (Drysdale (1983)). The metric in $\Re^3 \times SO(3)$ is non-obvious, and the CVD does not reduce to the GVD when rotations are factored out. However, it has the same connectivity as the GVD. The *CVD* for configuration spaces without a Euclidean distance metric is fundamentally different, and is defined as follows.

To define the CVD, we rely on the collection of pseudo-metrics provided by the geometric interpretation of C-function values (chapter 3). Intuitively, within some well-defined region in free-space where a C-function is non-redundant, its value characterizes the translational distance to either (1) an obstacle face, or (2) the plane of the obstacle face. Formally:

In this chapter, we will use $\mathcal{F} \subseteq \Re^3 \times SO(3)$ to denote free space. See chapter 2 for a formal review of charts and atlases. As noted in more detail in chapter 2, in this thesis we usually specify charts via the inverse form $h : E^n \rightarrow M$ (where E^n is an open subset of $\Re^n$) with the understanding that it is the inverse (or set of local inverses) h^{-1} which provides the family of charts $\{(h^{-1}, W_i)\}$, for $\bigcup_i W_i = h(E^n)$.

Definition (1): Let $\mathcal{M}$ be the set of families of C-functions on $\Re^3 \times SO(3)$. For $X \in \mathcal{F}$, let A_X be the set of maximum, applicable, non-redundant C-functions within families, that is, if $M \in \mathcal{M}$ is a family of C-functions, and $M' \subseteq M$ is the subset of applicable and non-redundant C-functions at X, then M contributes to A_X the function $f \in M'$ such that $f(X) > h(X)$ for all $h \in M' - \{f\}$. If k functions in M' tie for maximum, then M contributes all k to A_X.

Let n be the dimension of *C-Space*. Now, $X \in CVD$ if there exists a maximal subset B of A_X, containing at least two and no more than n C-functions, such that all functions in B have the same value b_0 at X and all functions in $A_X - B$ have a value greater than b_0. We say that $X \in CVD$ lies on an $(n - |B| + 1)$-dimensional *Voronoi Manifold.* The C-Voronoi Diagram for *C-Space* is the union of these Voronoi Manifolds.

We have seen that a level C-manifold is analogous to a level surface in $\Re^3$, in that it is the set of configurations $\{ X \mid f(X) = \ell \}$ for some applicable C-function f. Clearly, points on a k-dimensional Voronoi manifold $\mathcal{V}$ lie on the intersection of $n - k + 1$ *equal* level C-manifolds, i.e.,

$$f_1(X) = \cdots = f_{n-k+1}(X) = \ell(X)$$

where the level $\ell(X)$ is allowed to vary as X moves along V. Furthermore, we insist that the C-functions f_i constructing the Voronoi manifold must belong to pairwise distinct families.

When we say that a Voronoi manifold $\mathcal{V}_i$ is constructed from a set of constraints F_i, we mean that all the C-functions $f \in F_i$ have equal value along $\mathcal{V}_i$. An intersection manifold constructed from F_i is the intersection of level C-surfaces for constraints in F_i. By this we mean that first a level is chosen for each $f \in F_i$, and then the resulting level C-surfaces are intersected. In general, a level C-surface for a C-function f at level ℓ has the form

$$f^{-1}(\ell).$$

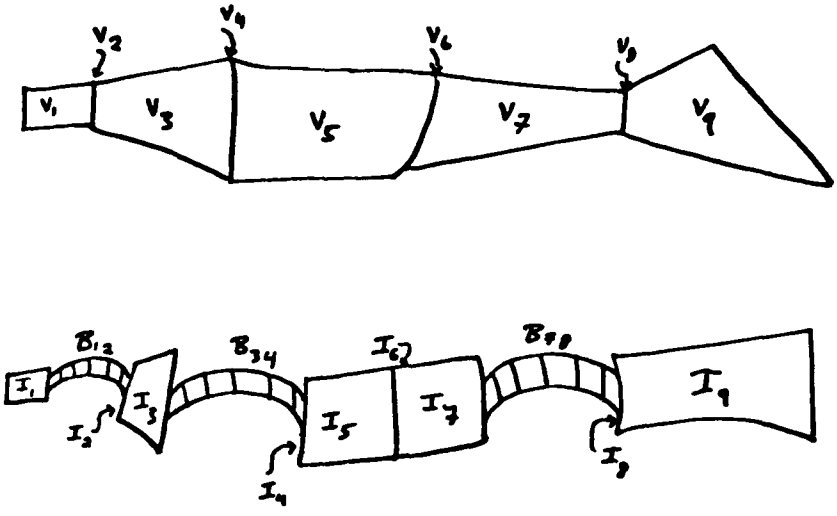


Figure 6.2. Strongly equivalent intersection manifolds, and the bridge manifolds connecting them. Each $\mathcal{V}_i$ is strongly equivalent to I_i. Each bridge manifold $B_{i,i+1}$ is equivalent (but not strongly equivalent) to $\mathcal{V}_i$.

$f^{-1}(\ell)$ is the *trivial* intersection manifold, that is, the manifold constructed by intersecting one level C-surface with itself. The intersection of two level C-surfaces is

$$I_i = f^{-1}(\ell_1) \bigcap g^{-1}(\ell_2).$$

The intersection manifold I_i' is constructed from the same C-functions at different levels:

$$I_i' = f^{-1}(\ell_1') \bigcap g^{-1}(\ell_2').$$

An Overview of the Proofs

Let p be a path along the CVD. p lies on a connected chain of Voronoi manifolds. Call this chain $\mathcal{V}_1, \ldots, \mathcal{V}_k$. For a Voronoi manifold $\mathcal{V}_i$ we say that an intersection manifold I_i is *equivalent* to $\mathcal{V}_i$ if (1) the set of C-functions which construct $\mathcal{V}_i$ is a (possibly non-strict) superset of the C-functions F_i' which define I_i, (2) I_i is homotopically equivalent to $\mathcal{V}_i$, (3) I_i lies in free space, and (4) all C-functions in F_i' satisfy definition (1) along I_i. If $F_i = F_i'$, i.e., I_i is constructed with exactly the same C-functions, then I_i is *strongly equivalent* to $\mathcal{V}_i$.

Note that by definition, each $\mathcal{V}_i$ is restricted to where the conditions of definition (1) hold, i.e., to some region $NR(F_i) \subset \mathcal{F}$ where all C-functions in F_i constructing $\mathcal{V}_i$ are maximum, applicable, non-redundant C-functions within families. Now, in general, I_i is an unbounded level set which cannot lie in free space everywhere. Thus by convention, we also restrict I_i to the region of interest $NR(F_i')$ where all C-functions in F_i' satisfy definition (1). That is, $NR(F_i') = \{ X \mid F_i' \subseteq A_X \}$. Instead of writing $I_i \cap NR(F_i')$ everywhere, this convention is assumed throughout.

In is an interesting question whether, for every Voronoi chain $\mathcal{V}_1, \ldots, \mathcal{V}_k$, there exists a connected, finite, corresponding equivalent or strongly equivalent chain of intersection manifolds $I_1, \ldots, I_{k'}$ (where k is not necessarily equal to k'). Theorem (I) shows that for all Voronoi manifolds $\mathcal{V}_i$, there exists a strongly equivalent intersection manifold I_i. These I_i might not form a connected chain. Theorems (II) and (III) show that each disconnected pair of intersection manifolds I_i and I_{i+1} can be connected by an infinite sequence of "bridges." Each bridge is an intersection manifold equivalent (but not strongly equivalent) to $\mathcal{V}_i$. We then argue that since there exists an infinite bridge sequence, therefore there must also exist a finite bridge sequence. Finally, (theorem IV) we show that there exists an *entire* intersection chain

$$I_1 \bigcup \cdots \bigcup I_{k'}$$

which is homotopically equivalent to the entire Voronoi chain

$$\mathcal{V}_1 \bigcup \cdots \bigcup \mathcal{V}_k.$$

Theorem I: Let $\mathcal{V}_k$ be an m-dimensional Voronoi manifold, constructed from a set of applicable, non-redundant constraints A, which satisfy definition (1) along $\mathcal{V}_k$. Then if there exists an intersection manifold I_k of level C-surfaces for the constraints A, and if the constraints A satisfy definition (1) along I_k, then $\mathcal{V}_k$ and I_k are homotopically equivalent.

Proof: We will show that two intersection manifolds constructed from the same C-functions at different levels are homotopically equivalent. Next, we demonstrate that the Voronoi manifold is essentially a special case of intersection manifold.

Let $\mathcal{V}_k$ be an m-dimensional Voronoi manifold,

$$\mathcal{V}_k = \{ X \mid f_1(X) = \cdots = f_n(X) \}, \qquad (n = 6 - m + 1)$$

where the f_i are chosen from A as defined above (see definition (1)). Note that the value of the f_i may vary with $X \in \mathcal{V}_k$. Let I_k be a $(m-1)$-dimensional (or 0-dimensional, if $m = 0$) intersection manifold of level C-surfaces constructed from the same functions f_i:

$$I_k = \{ X \mid f_1(X) = \ell_1, f_2(X) = \ell_2, \ldots, f_n(X) = \ell_n \}.$$

The region of interest for $\mathcal{V}_k$ and I_k is of course restricted to $NR(\{ f_i \})$, where the functions $\{ f_i \}$ satisfy definition (1). I_k differs from $\mathcal{V}_k$ in that on $\mathcal{V}_k$ the values (levels) of the functions f_i are equal, whereas on I_k, they are not. Furthermore, on $\mathcal{V}_k$ the value varies, whereas on I_k the values are fixed.

I_k may be expressed as

$$I_k = \bigcap_i f_i^{-1}(\ell_i).$$

The claim is that I_k is homotopically equivalent to $\mathcal{V}_k$, that is, that if $g : E^k \rightarrow \mathcal{F}$ is a chart for I_k and $g' : E^k \rightarrow \mathcal{F}$ is a chart for $\mathcal{V}_k$, then there exists a continuous homotopy deformation $h : E^k \times I^1 \rightarrow \mathcal{F}$ between g and g' such that

$$h(Y,0) = g(Y)$$
$$h(Y,1) = g'(Y).$$

As usual, I^1 denotes the unit interval $[0,1]$. For a review of elementary homotopy theory consult appendix II.

The charts g and g' exist, since I_k and $\mathcal{V}_k$ are manifolds. (Assume without loss of generality that only one chart is required). A level C-manifold $f_i^{-1}(\ell)$ (for some level ℓ) is a 5-dimensional manifold and hence there exists a chart $C_i : E^5 \to \mathcal{F}$ for $f_i^{-1}(\ell)$. We demonstrate such charts in the proof of claim (I.1), below.

Let $\ell_\mathcal{V}$ be any achievable value for the functions f_i along the Voronoi manifold $\mathcal{V}_k$, that is, any $\ell_\mathcal{V}$ such that there exists some $X \in \mathcal{V}_k$ satisfying definition (1) for which

$$f_1(X) = \cdots = f_n(X) = \ell_\mathcal{V}.$$

Now, $t\ell_i + (1-t)\ell_\mathcal{V}$ is a linear combination of the levels ℓ_i and $\ell_\mathcal{V}$ for f_i. Since each level C-surface

$$f_i^{-1}\Big(t\ell_i + (1-t)\ell_\mathcal{V}\Big)$$

is a manifold, each has a chart of the form C_i, above. If $t \in [0,1]$, these are C-surfaces for f_i with level $\ell \in [\ell_\mathcal{V}, \ell_i]$, and their charts may be parameterized[1] by t. Suppose we have a set of several level C-surfaces (as in I_k). Their charts may be intersected to form a new chart for the intersection manifold. We define $h : E^k \times I^1 \to \mathcal{F}$ to be the chart for the intersection manifold at t, such that

$$h(E^k, t) = \bigcap_i f_i^{-1}\Big(t\ell_i + (1-t)\ell_\mathcal{V}\Big).$$

We call h a *chart family* for the intersection manifold.

Claim (I.1): The chart family h can be constructed such that $h(Y,t)$ is continuous in Y and t, within the area of interest for $\mathcal{V}_k$ and I_k. (For proof, see below).

[1] See the proof of claim (I.1), where h_C is such a chart.

Recall that ℓ_i is the value (or level) of f_i on the intersection manifold I_k. Then $h : E^k \times I^1 \rightarrow \mathcal{F}$ is a homotopy between g and g', that is, h continuously deforms I_k into $\mathcal{V}_k(\ell_\mathcal{V})$, where we use $\mathcal{V}_k(\ell_\mathcal{V}) \subseteq \mathcal{V}_k$ to denote the Voronoi manifold *restricted to level* $\ell_\mathcal{V}$, i.e.,

$$\mathcal{V}_k(\ell_\mathcal{V}) = \{ X \mid f_1(X) = \cdots = f_n(X) = \ell_\mathcal{V} \} \subset \mathcal{F}.$$

Verify that

$$\begin{aligned} h(E^k, 1) &= \bigcap_i f_i^{-1}(\ell_i) \\ &= I_k, \end{aligned}$$

and that

$$\begin{aligned} h(E^k, 0) &= \bigcap_i f_i^{-1}(\ell_\mathcal{V}) \\ &= \mathcal{V}_k(\ell_\mathcal{V}). \end{aligned}$$

We have shown that $I_k \simeq_h \mathcal{V}_k(\ell_\mathcal{V})$ ($\simeq_h$ denotes homotopic equivalence) for all achievable $\ell_\mathcal{V}$. We must now show that $\mathcal{V}_k(\ell_\mathcal{V}) \simeq_h \mathcal{V}_k(\ell_\mathcal{V} + \epsilon)$.

We are interested in continuous deformation within $\mathcal{F}$. Hence $\mathcal{V}_k$ may be multiply connected within $\mathcal{F}$, so long as it does not wrap around obstacles. $\mathcal{V}_k$ must be contractible to a point (within $\mathcal{F}$). This is guaranteed by the construction of A (definition (1)) and $\mathcal{V}_k$, i.e., by the choice and domains of the functions f_i. To see this, consider that if $\mathcal{V}_k$ *did* wrap around an obstacle in *C-Space*, then the value of some f_i would have to go negative. Hence, it would become redundant, and could not be used in the construction of $\mathcal{V}_k$. Note that the C-Voronoi *diagram*, which is the union of Voronoi *manifolds* such as $\mathcal{V}_k$, will, in general, wrap around obstacles and be multiply connected.

Furthermore, we can choose ϵ such that the topology of $\mathcal{V}_k$ does not change too drastically between $\mathcal{V}_k(\ell_\mathcal{V})$ and $\mathcal{V}_k(\ell_\mathcal{V} + \epsilon)$. (This is possible since $\mathcal{V}_k$ is finite-branching). So

$$I^k \simeq_h \mathcal{V}_k(\ell_\mathcal{V}) \simeq_h \mathcal{V}_k(\ell_\mathcal{V} + \epsilon_1) \simeq_h \mathcal{V}_k(\ell_\mathcal{V} + \epsilon_2) \simeq_h \cdots \simeq_h \mathcal{V}_k(\ell_\mathcal{V} + \epsilon_r).$$

What we have shown is that I_k is homotopically equivalent to the "easy" parts of $\mathcal{V}_k$ (where the level of the Voronoi manifold is constant). We next showed that because the topology of $\mathcal{V}_k$ is simple, we can paste together these restricted Voronoi manifolds. ∎

Proof of Claim I.1: The existence of a continuous chart family for the intersection manifold is based on our knowledge that the manifolds exist at certain levels, and from our ability to demonstrate such a chart for the intersection manifold. In chapter 4, we exhibited C-functions of six variables for the *C-Space* $\Re^3 \times SO(3)$:

$$f_i : (x, y, z, \psi, \theta, \phi) \rightarrow \Re$$

which are continuous, affine in x, y, and z, and multilinear in the sines and cosines of the angles ψ, θ, and ϕ. The *Linear Form* for a C-function $f_i : \Re^3 \times SO(3) \rightarrow \Re$, is an equivalent expression

$$f_i(x, y, z, \Theta) = E_1(\Theta)x + E_2(\Theta)y + E_3(\Theta)z + E_4(\Theta), \tag{6.1}$$

where $E_j : SO(3) \rightarrow \Re$ (for $j = 1, 2, 3, 4$). Now,

$$f_i^{-1}(\ell) = \ker\Big(f(x, y, z, \psi, \theta, \phi) - \ell\Big).$$

A chart for $f_i^{-1}(\ell)$ may be found by solving

$$f_i(x, y, z, \psi, \theta, \phi) - \ell = 0$$

for x (or y, or z) in terms of the other variables and ℓ:

$$x = -\frac{E_2 y + E_3 z + E_4 - \ell}{E_1}.$$

(We have dropped the Θ, since the functions E_j are constant with respect to x, y, z, and ℓ). If $E_1 = 0$, then the solution for y or z may be employed. This yields the obvious chart

$$\begin{aligned} C : E^5 &\to \Re^3 \times SO(3) \\ (y, z, \psi, \theta, \phi) &\mapsto (-\frac{E_2 y + E_3 z + E_4 - \ell}{E_1}, y, z, \psi, \theta, \phi) \end{aligned}$$

which we presented in chapter 4. C is affine in ℓ, and can be used to construct a family of charts

$$h_C : (Y, \ell) \to \Re^3 \times SO(3)$$

for the C-surfaces

$$f_i^{-1}(\ell) = f_i^{-1}\Big(t\ell_i + (1-t)\ell_\nu\Big).$$

which is continuous in $Y = (y, z, \psi, \theta, \phi)$ and ℓ. h_C is clearly a homotopy between level C-manifolds for f_i.

Chapter 4 also derives charts for $f_i^{-1}(\ell_i) \cap f_j^{-1}(\ell_j)$, $f_i^{-1}(\ell_i) \cap f_j^{-1}(\ell_j) \cap f_k^{-1}(\ell_k)$, (and so on) by solving the C-functions simultaneously for the intersection manifold. For example, a chart for the intersection manifold of degree three

$$\bigcap_{i=1,2,3} f_i^{-1}(\ell_i)$$

can be constructed by solving three simultaneous equations with the form of (6.1). For arbitrary coefficients E_j and levels ℓ_i, this intersection may not always exist. However, we know *a priori* that it exists for the specified levels ℓ_i and ℓ_ν. From the form of (6.1), it is clear that if the intersection manifold exists for some levels

ℓ_i (and if the coefficient functions E_j are independent), then it will exist for all levels. We omit a discussion of intersection manifolds of higher degree: the reader is referred to chapter 4 for further details. ∎

Corollary: For every Voronoi manifold $\mathcal{V}_i$, there exists a strongly equivalent intersection manifold I_i.

Corollary: If F_i is the set of constraints used to construct $\mathcal{V}_i$, let $NR(F_i)$ denote the region in free-space where all of the constraints in F_i satisfy definition (1). If $\mathcal{V}_i$ exists, then in every connected component of $NR(F_i)$, there exists a strongly equivalent intersection manifold I_i built out of F_i.

Proof of corollaries: All $f \in F_i$ exist within $NR(F_i)$. Pick any $X \in NR(F_i)$. Evaluate all the functions in F_i at X to obtain a set of levels. The intersection manifold must exist at these levels, since we have demonstrated that X is on the intersection manifold. The intersection manifold from the C-surfaces at these levels is by definition and by theorem (I) strongly equivalent to $\mathcal{V}_i$. ∎

Next, observe that for all $1 < i < k$, either $\mathcal{V}_i \subset \mathcal{V}_{i+1}$ or $\mathcal{V}_{i+1} \subset \mathcal{V}_i$. In other words, to move from $\mathcal{V}_i$ to $\mathcal{V}_{i+1}$, we either add or remove one or more constraints:

$$\mathcal{V}_i = \{ X \mid f_1(X) = \cdots = f_{k_i} \}$$
$$\mathcal{V}_{i+1} = \{ X \mid f_1(X) = \cdots = f_{k_{i+1}} \}$$

and either $k_i > k_{i+1}$ or $k_i < k_{i+1}$. We call k_i and k_{i+1} the *degree* of the Voronoi manifolds.

We have shown that for a Voronoi chain $\mathcal{V}_1, \ldots, \mathcal{V}_k$, a sequence of intersection manifolds $I_1, \ldots, I_k$ may be constructed such that each I_i is strongly equivalent to $\mathcal{V}_i$ (for $1 \leq i \leq k$). However, the sequence of intersection manifolds may be disconnected. We now furnish a theorem demonstrating that the intersection manifolds may be constructed in such a manner that they can be connected together by a series of special intersection manifolds, called bridges.

Theorem II: The intersection manifolds $I_1, I_2, \ldots, I_k$ may be constructed such that each pair of intersection manifolds I_i and I_{i+1} can be connected by a sequence

of "bridges." Each bridge is an intersection manifold equivalent (but not strongly equivalent) to $\mathcal{V}_i$.

Proof: Let I_1 be a strongly equivalent intersection manifold to $\mathcal{V}_1$, constructed with C-functions F_1. Along I_1, all constraints in F_1 are non-redundant. Let $\mathcal{V}_2$ be the next Voronoi manifold after $\mathcal{V}_1$ in the Voronoi chain, and let F_2 be the constraints constructing $\mathcal{V}_2$.

Case (i): If $F_2 \subset F_1$, then $\mathcal{V}_1$ is lower in dimension that $\mathcal{V}_2$. We can construct I_2, a strongly equivalent intersection manifold to $\mathcal{V}_2$, which is connected with I_1, be removing one or more constraints in F_1. (We remove exactly the constraints $F_1 - F_2$). This is possible because if $F_2 \subset F_1$, then $NR(F_1) \subset NR(F_2)$:

$$I_1 = \bigcap_{f_j \in F_1} f_j^{-1}(c_j)$$
$$I_2 = \bigcap_{f_j \in F_2} f_j^{-1}(c_j). \tag{6.2}$$

Strictly speaking, equation (6.2) should employ the subset notation ($\subset$) instead of equality ($=$), since I_1 and I_2 are restricted to where the intersection is applicable and non-redundant. However, the equality makes the construction more transparent. Note that the construction still works with the subset notation, since $NR(F_1)$ is a subset of $NR(F_2)$. Since $F_2 \subset F_1$, I_1 and I_2 agree on the levels for C-functions in F_2. Since $I_1 \subset I_2$, I_1 and I_2 are connected.

Case (ii-a): Suppose, however, that $F_1 \subset F_2$. Then $\mathcal{V}_2$ is lower in dimension than $\mathcal{V}_1$, and $NR(F_2) \subset NR(F_1)$.

We know that $I_1 \subset NR(F_1)$. If $I_1 \cap NR(F_2) \neq \emptyset$, then we can construct I_2 from F_2 such that $I_2 \subset NR(F_2)$, $I_2 \subset I_1$, and in addition, I_1 and I_2 agree on the levels in F_1. Construct I_2 as follows: pick a point $X_0 \in I_1 \cap NR(F_2)$. Evaluate each $f_j \in F_2 - F_1$ at X_0, to obtain a level $c_j = f_j(X_0)$. Construct:

$$I_2 = I_1 \bigcap \Big(\bigcap_{f_j \in F_2 - F_1} f_j^{-1}(c_j) \Big).$$

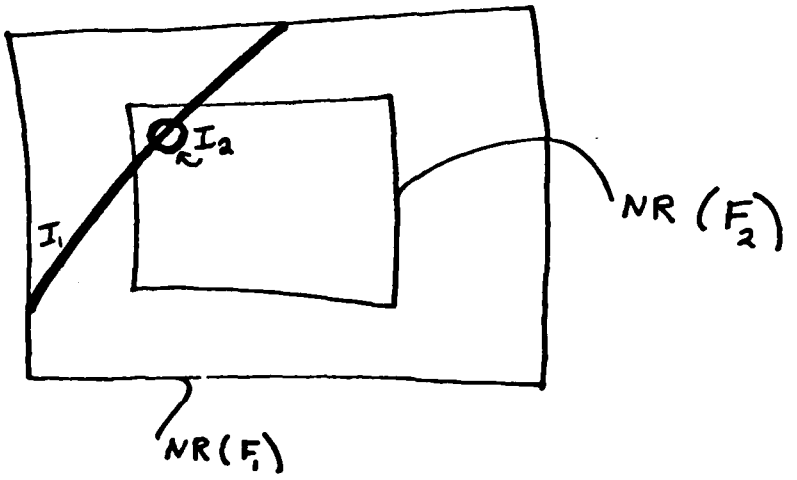


Figure 6.3. Case where $I_1 \bigcap NR(F_2) \neq \emptyset$.

We showed earlier that I_2 may be constructed in this manner.

Case (ii-b): The hard case is when $I_1 \bigcap NR(F_2) = \emptyset$. In this case, we must construct some strongly equivalent (to $\mathcal{V}_2$) intersection manifold $I_2 \subset NR(F_2)$ with different levels from I_1 with respect to the C-functions F_1. We then build a sequence of *bridge manifolds*, entirely within $NR(F_1)$, between I_1 and I_2, connecting them together.

The bridge intersection manifolds are constructed out of some subset $F_B \subset F_1$, and each bridge manifold is equivalent (but not strongly equivalent) to $\mathcal{V}_1$. The bridge manifolds are formed by relaxing one or more constraints in F_1 to be able to move from $NR(F_1) - NR(F_2)$ into $NR(F_2)$. The motions slide along intersection manifolds constructed from the remaining constraints. Once inside $NR(F_2)$, we construct I_2 there, and I_2 is strongly equivalent to $\mathcal{V}_2$. Note the levels at which I_2 is constructed, (with respect to the constraints in F_1), are typically different from

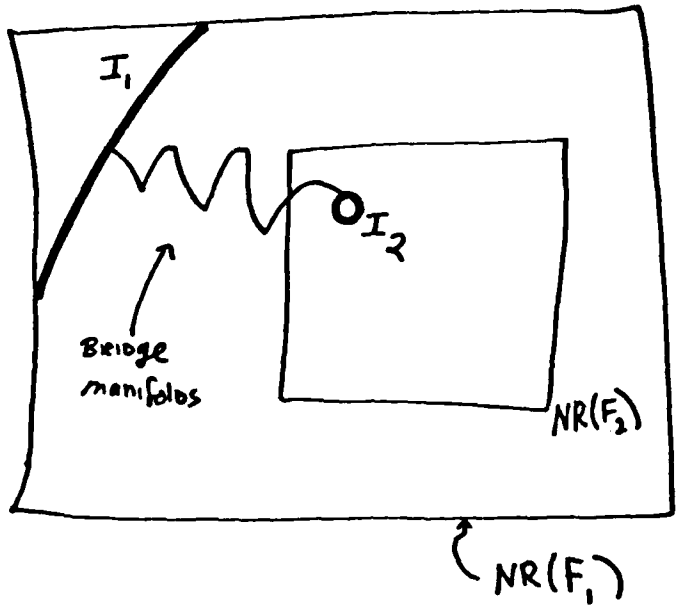


Figure 6.4. Case where $I_1 \bigcap NR(F_2) = \emptyset$.

the levels at which I_1 is constructed. The existence of bridge manifolds is covered in theorem III. ∎

We will use the dot notation for the Riemannian inner product on the tangent space at X. The normal to a level C-surface $f^{-1}(\ell)$ is ∇f, and depends on the inner product. We assume throughout that the normals are unit vectors. A basic concept in these proofs is that of general position, or *transversality* (see Hirsch, 1976). Two submanifolds M, N of a manifold V are in general position if at every point of $M \bigcap N$ the tangent spaces of M and N span that of V. If A and B are not in general position, then arbitrarily small perturbations of one of them will put them in general position. In our case, M and N correspond to level C-surfaces and their intersection manifolds, and V to $\Re^3 \times SO(3)$. The proofs still work even if M and N are in general position only at "many" points of $M \bigcap N$. We will write the condition of general position for two level C-surfaces $f^{-1}(\ell_1)$ and $g^{-1}(\ell_2)$ as $\nabla f \cdot \nabla g \neq 1$.

Theorem III *The Existence of Bridge Manifolds:* (Bridges of dimension five). Let S be a path-connected component of $NR(F_1)$, $X_0, X_1 \in S$, and $f, g \in F_1$. Note that S lies with in the domain of f and g, and that by construction, $i(S)$ is an open set. Then if $\nabla f \cdot \nabla g \neq 1$ within S, then there exists an infinite sequence of five dimensional bridge manifolds within S, connecting X_0 and X_1.

Proof: Each bridge manifold will be of the form $f^{-1}(c_f)$ or $g^{-1}(c_g)$, for different levels of c_f and c_g. Note that as we vary c_f, $f^{-1}(c_f)$ covers S (similarly for g). Let T_X denote the six dimensional tangent space at X. If $c_0 = f(X)$, the level C-surface $f^{-1}(c_0)$ is a five dimensional submanifold of $\Re^3 \times SO(3)$, with a five dimensional tangent space, T_X^f. That is, identifying T_X^f with a subspace of T_X,

$$T_X^f = \{ v \in T_X \mid v \cdot \nabla f(X) = 0 \}.$$

It is easy to show that $T_X^f \cup T_X^g$ spans T_X, for all X where $\nabla f(X) \cdot \nabla g(X) \neq 1$.

Let N_X^f denote the normal space at X with respect to f, such that

$$N_X^f = \{ v \in T_X \mid v = \alpha \nabla f(X) \},$$

for all scalars $\alpha \in \Re$. So $T_X = N_X^f \oplus T_X^f$. Clearly, if $\nabla f(X) \cdot \nabla g(X) \neq 1$, then N_x^f is spanned by $T_X^f \cup T_X^g$. Hence $T_X = T_X^f + T_X^g$.

Since the space of differentially tangent directions to the two level C-surfaces at X is equal to the space of all directions, there exists an infinite sequence of differential moves along level C-surfaces for f and g, at different levels, to realize any path within S. Since S is connected, there exists such a path from X_0 to X_1. ∎

Corollary III.1: (Bridges of dimension four). A direct result is the existence of a sequence of bridges which are four dimensional intersection manifolds. Let f_1, f_2, f_3, $f_4 \in F_1$. Suppose that within S, $\nabla f_i(X) \cdot \nabla f_j(X) \neq 1$ (for $i \neq j$). Then there exists an infinite sequence of bridges between X_0 and X_1, where each bridge is of the form

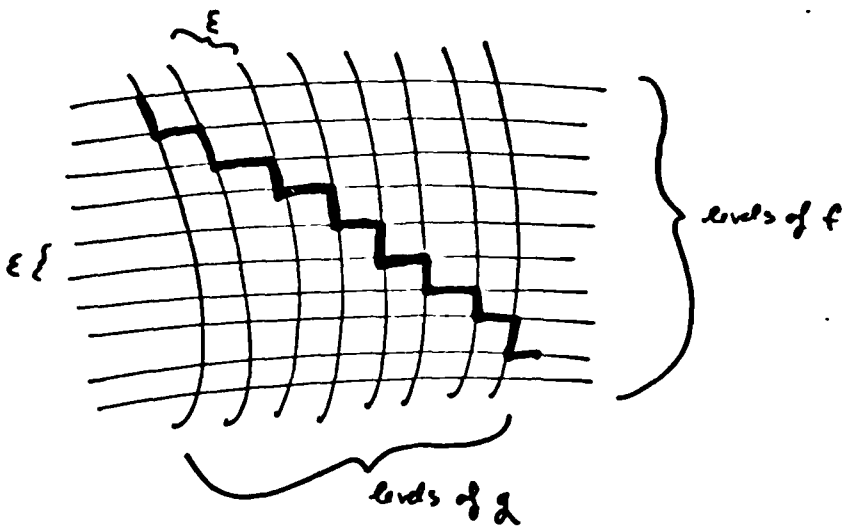


Figure 6.5. A finite path iterating along levels of f and levels of g.

$$I_B(i,j) = f_i^{-1}(c_i) \bigcap f_j^{-1}(c_j). \qquad (i \neq j)$$

This is reasonable, since each $I_B(i,j)$ is a four dimensional manifold. If the normals are all independent at X, then the direct sum of the tangent spaces to all possible intersection manifolds $I_B(i,j)$ is clearly T_X:

$$T_X = (T_X^{f_1} \bigcap T_X^{f_2}) + (T_X^{f_1} \bigcap T_X^{f_3}) + \cdots + (T_X^{f_i} \bigcap T_X^{f_j}) + \cdots + (T_X^{f_3} \bigcap T_X^{f_4}).$$

(Of course, $i \neq j$ for all terms in this sum). ∎

Corollary III.2: (Existence of a finite sequence of bridges). We now argue that if there exists an infinite sequence of bridges from X_0 to X_1 within S, then there also exists a finite sequence.

Informally, we argue that it is always possible to to move a certain distance ϵ along each level C-surface, and that this ϵ cannot grow arbitrarily small. First of all, note that S is not infinitesimal, and that $i(S)$ is an open set. (If it were not, it might be necessary to make an infinite number of differential motions to remain within S).

We also appeal to the well-behaved structure of the level C-surfaces, and their intersection manifolds. The level C-surfaces are smooth, with normals that change continuously. (If the normals changed discontinuously, we might not be able to take finite steps). Thus we can move a finite (i.e., not infinitessimal) distance along the surfaces to a point where the normals are still independent, and where the surfaces are "similar" (i.e., having normals in almost the same direction as before). Furthermore, for any two levels of f within S, there exists a homotopy between them. These continuity arguments indicate that it should be possible to move in finite steps along the intersection manifolds, and hence we can reach X_1 from X_0 in a finite number of bridges.

Suppose from X_0 to X_1 there exists an infinite sequence of bridge manifolds, but no finite sequence. Then either (1) $i(S)$ is not an open set (and therefore only differential motions can stay within it), or (2) for a subset $P \subset S$, whose cardinality is that of the continuum, the entire tangent space is not available along the level C-surfaces. In both cases, our initial hypotheses are violated. (1) violates the assumption $i(S)$ is an open set, and (2) the assumption of general position.

We formalize this argument as follows:

Definition: Let $\mathcal{U}$ be a metric space, and $p, p' : I^1 \rightarrow \mathcal{U}$ be paths in $\mathcal{U}$. Let $\{U_\alpha\}$ be an open cover of $p(I^1)$ in $\mathcal{U}$, where each U_α is a neighborhood of radius $\leq r$, and $U_\alpha \bigcap p(I^1) \neq \emptyset$. We say that p' *approximates* p *at resolution* r if $\{U_\alpha\}$ is an open cover for $p'(I^1)$ also; that is, if $p'(I^1) \subset \bigcup_\alpha U_\alpha$.

Claim (III.2.1) shows that an arbitrary curve in some neighborhood U of free-space can be approximated by a path within U along a finite sequence of intersection manifolds. The proof of Cor. (III.2) then employs the fact that the curve is compact, and therefore can be covered by a finite number of such neighborhoods.

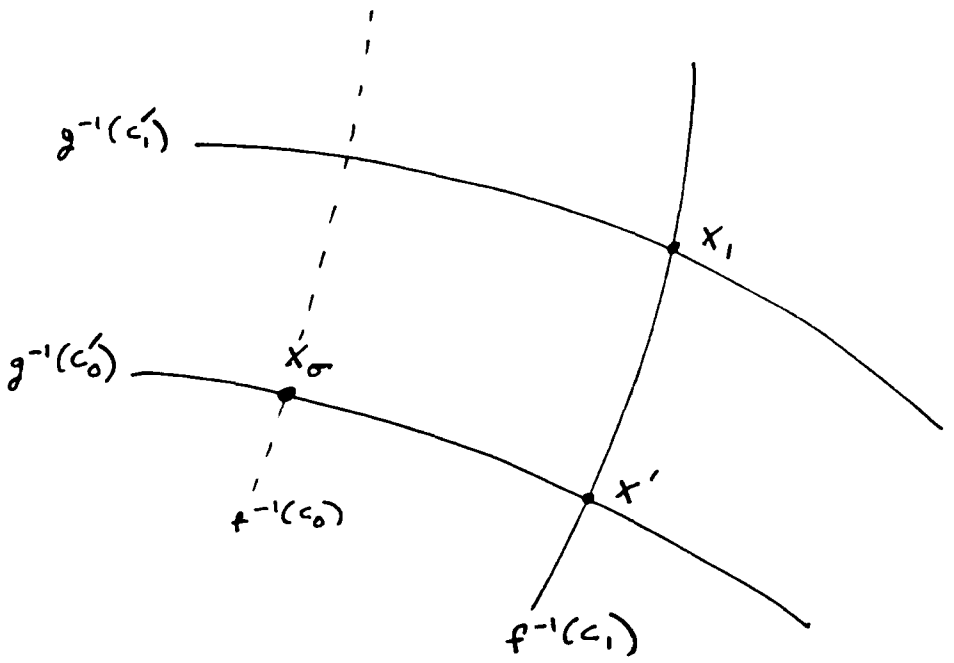


Figure 6.6. Let $c_0 = f(X_0)$ and $c_1 = f(X_1)$. X_0 lies on $f^{-1}(c_0) \bigcap g^{-1}(c_0')$, and X_1 lies on $f^{-1}(c_1) \bigcap g^{-1}(c_1')$. The path segment between X_0 and X_1 may be approximated by a path sliding first along $g^{-1}(c_0')$ from X_0 to X', and then from X' to X_1 along $f^{-1}(c_1)$.

Claim III.2.1: Any path within a neighborhood where f and g are defined, may be approximated to an arbitrary resolution by a finite sequence of motions along level C-surfaces of f and g.

Proof: We will regard level C-surfaces of f and g as trivial intersection manifolds. Consider how one can approximate a path from X_0 to X_1 by a path along intersection manifolds. Let d be a metric on $\Re^3 \times SO(3)$, and f, g be C-functions with inverse images covering a neighborhood containing X_0 and X_1. Let $c_0 = f(X_0)$ and $c_1 = f(X_1)$.* X_0 lies on the intersection manifold $f^{-1}(c_0) \bigcap g^{-1}(c_0')$, and X_1 lies on $f^{-1}(c_1) \bigcap g^{-1}(c_1')$. Construct $f^{-1}(c_1) \bigcap g^{-1}(c_0')$ (refer to fig. 6.6), and choose X' to be the any of the closest points to X_0 on this manifold. We can construct a path which slides from X_0 to X' along $g^{-1}(c_0')$, and then slides from X' to X_1 along $f^{-1}(c_1)$. We wish to demonstrate that by choosing X_1 sufficiently close to X_0, X' can be made to lie arbitrarily close to X_0; that is, for all $\epsilon > 0$, there exists a

* and $c_0' = g(X_0)$, $c_1' = g(X_1)$.

$\delta > 0$ so that $d(X_0, X') < \epsilon$ whenever $d(X_0, X_1) < \delta$. This is definitional, since $\lim_{X_1 \to X_0} f^{-1}(c_1) = f^{-1}(c_0)$.

Thus for every neighborhood $B_r(X_0)$ of radius r about X_0, there exists an $X_1 \in B_r(X_0)$ such that $X' \in B_r(X_0)$ also. Furthermore, $d(X_0, X')$ is finite (and non-zero). Of course, a similar argument holds for the path segment between X_1 and X'. Thus any path within a neighborhood where f and g are defined, may be approximated to an arbitrary resolution by a finite sequence of motions along intersection manifolds. ∎

Clearly, similar arguments hold for intersection manifolds of higher degree.

Proof (Corollary III.2): (Sketch) Let $p(I^1)$ be a path within $i(S)$ from X_0 to X_1 along an infinite sequence of intersection manifolds. In short, $p(I^1) \subset i(S)$. Choose an open cover $\{U_\alpha\}$, relative to $\Re^3 \times SO(3)$, for $p(I^1)$ such that $\bigcup_\alpha U_\alpha \subset i(S)$. $p(I^1)$ is compact, hence there exists a finite subcover, i.e., for finitely many indices $\alpha_1, \ldots, \alpha_n$, we have

$$p(I^1) \subset U_{\alpha_1} \bigcup \cdots \bigcup U_{\alpha_n} \subset i(S).$$

Now, for each U_{α_i}, we can construct a path $p'(I^1)$ along a finite sequence of intersection manifolds approximating $p(I^1) \bigcap U_{\alpha_i}$ (Claim III.2.1). That is, $p'(I^1)$ is also contained within the closure of U_{α_i}. Furthermore, it is not hard to construct p' such that it leaves the neighborhood U_{α_i} at the same points as p, that is, so that $p(I^1) \bigcap \partial U_{\alpha_i} = p'(I^1) \bigcap \partial U_{\alpha_i}$. Since this is true for all U_{α_i}, then $p(I^1)$ can be approximated by some path contained within $U_{\alpha_1} \bigcup \cdots \bigcup U_{\alpha_n}$, and which lies on some finite chain of intersection manifolds. ∎

Theorem (IV) is almost immediate, and its proof similar to that of Theorem (I).

Theorem IV: For every connected chain of Voronoi manifolds $C_\mathcal{V} = \mathcal{V}_1, \ldots, \mathcal{V}_k$, there exists an equivalent (in the sense of theorems (I—III)) connected chain of intersection manifolds $C_I = I_1, \ldots, I_k$, such that the entire Voronoi chain $C_\mathcal{V}$ is homotopically equivalent to the intersection chain C_I. That is,

$$\bigcup_{\mathcal{V}_i \in C_\mathcal{V}} \mathcal{V}_i \simeq_h \bigcup_{I_j \in C_I} I_j.$$

Proof: Simply apply the fact that each equivalent pair $(\mathcal{V}, I)$ (where $\mathcal{V}$ is a Voronoi manifold and I is an equivalent intersection manifold) must lie in the same non-redundancy region. Hence if one chain wraps around an obstacle, so must the other; furthermore, each chain must wrap around the same obstacles. ∎

Future Research

There are several interesting problems which are left open. They include the following:

(i) We have demonstrated an equivalent chain of intersection manifolds for any connected chain of Voronoi manifolds. Show whether or not a strongly equivalent chain exists also.

(ii) Show whether or not for every Voronoi chain $C_\mathcal{V}$ there exists a (strongly) equivalent intersection chain exhibiting a bijective correspondence to $C_\mathcal{V}$.

(iii) Devise an effective procedure for constructing a chain of intersection manifolds to realize any class of paths in free space.

(iv) Derive complexity bounds on the construction of the C-Voronoi diagram and the equivalent intersection chains.

(v) Other configuration spaces, such as those arising in the hinged body problem, should be considered.

(vi) To extend these results to configuration spaces generated by real space constraints which are not polyhedral (for example, algebraic surfaces), a generalization of the GVD such as smoothed local symmetries (Brady, 1982b) could be considered.

7
Conclusion

In chapters 1 through 6, we developed representations and algorithms for automated spatial planning with six degrees of freedom. To demonstrate the competence of the representation and the feasibility of the algorithms, a planning system for the classical *find-path problem with six degrees of freedom* was implemented. The planner is of considerable intrinsic interest, in that it is complete (for a given resolution). Experiments have demonstrated that this algorithm can solve find-path problems requiring six degree of freedom solutions that were beyond the competence of earlier, approximate planners. The mathematical framework developed here impacts a class of geometric planning problems for three dimensional objects.

The planning algorithm may be explained by analogy with the Point Navigation Operators. The *C-Space* transformation reduced the motion planning problem to the task of navigating a point in $\Re^3 \times SO(3)$. Since the path for the point must avoid the *C-Space* obstacles, which are curved, six dimensional manifolds with boundary, clearly paths can be found in *C-Space* by the closure of three operators:

(i) slides along 1- to 4-dimensional intersections of level C-surfaces;

(ii) slides along 5-dimensional level C-surfaces;

(iii) jumps between 6-dimensional obstacles.

However, these simple operators could not be implemented until a series of representational and algorithmic questions were solved. The fundamental representational issues centered on the structural properties of the defining C-functions. By deriving their domains, and by proving theorems about the topology of these domains, it was then possible to address the intersection problem for high-dimensional configuration spaces. By solving these open problems, developing representations for the C-functions and their domains, and designing decomposition algorithms in *C-Space*, it became possible to represent the constraints on motion "completely," and to exploit the complete representation in implementing a planning algorithm. Next, we presented new theoretical results on the C-Voronoi diagram. By showing that for every connected chain of Voronoi manifolds, there exists an equivalent chain of intersection manifolds of level C-surfaces, a theoretical completeness result for planning along the intersection manifolds was obtained. This result is also of interest since while the charts for the Voronoi manifolds are as yet unknown, charts for the intersection manifolds are straightforward (given our representational framework). Thus it is possible, in principle, to devise a planning algorithm with all the advantages of a *planner* along the CVD.

There is much work to be done. Ultimately, decomposition algorithms such as those we present in chapter 5 will become increasingly important in partitioning *C-Space* into regions where the set of applicable (or alternatively, relevant) constraints is invariant (see also Schwartz and Sharir (1981)). The representations and algorithms we have developed may make other geometric planning problems--such as fine-motion, and planning with uncertainty--more feasible, and should now be applied in these applications. The find-path algorithm can be easily extended to robot manipulators with six degrees of freedom in which translations can be decoupled from rotations. This class includes Cartesian manipulators (for example, the IBM RS/1). The adaptation of this work to a production environment presents interesting engineering challenges.

In principle, the 6DOF representations can be extended to revolute-joint, linked arms with six degrees of freedom. However, the *C-Space* of the linked-arm problem is the six dimensional torus,

$$S^1 \times \cdots \times S^1 \qquad \text{(to 6)}$$

which has a very different structure from $\Re^3 \times SO(3)$. It is our hope that this thesis can present a methodology for formulating the geometric constraints for arbitrary configuration spaces, and that a similar structure will be found for constraints on the 6-torus.

Our planning algorithm is complete (at a resolution), in that the representation employed is complete, and in that the search is guaranteed to find a path if one exists at that resolution. However, since it is a search algorithm, we cannot provide a polynomial time bound. Our motivation has been to address the completeness issue first, by resolving fundamental representational questions; now, one of the most important remaining tasks is to develop complete, polynomial-time algorithms which can actually be implemented. We do not believe that the worst-case exponential behavior of the 6DOF planner is inherent in the representation, and conjecture that a polynomial-time algorithm which plans paths along intersection manifolds can be devised. Indeed, the theoretical results on the CVD are suggestive that the limiting complexity of the approach may be the complexity of constructing the CVD or an equivalent chain of intersection manifolds. More research is needed on the topology of the CVD. A fast planning system might determine what constraints construct the CVD, and, using these constraints, construct a chain of intersection manifolds which could attain the goal. The first step in this effort would bound the complexity of the CVD and the intersection chains.

Appendix I

Details of the Intersection Problem, and Related Problems

This appendix contains the detailed equations from chapter 4, which are relegated to an appendix because of their excessive length. Some definition of terms is repeated, so that the interested reader will not have to flip back and forth too much.

I.1. Intersecting Two C-surfaces in $\Re^2 \times S^1$

By applying trigonometric reductions we can express type (a) and (b) constraints as follows (only (a1) and (b1) are shown):

$$\begin{aligned} &\cos(\lambda_i)Sy + C\sin(\lambda_i)y - \sin(\lambda_i)Sx + C\cos(\lambda_i)x \\ &\quad + \sin(\lambda_i - \gamma_j)\|b_j\|S - \|a_i\|\cos(\lambda_i - \eta_i) \\ &\quad - C\cos(\lambda_i - \gamma_j)\|b_j\| \end{aligned} \tag{a1}$$

$$\begin{aligned} &\sin(\phi_j)y + \cos(\phi_j)x - \|a_i\|\sin(\phi_j - \eta_i)S \\ &\quad - C\|a_i\|\cos(\phi_j - \eta_i) - \|b_j\|\cos(\phi_j - \gamma_j) \end{aligned} \tag{b1}$$

Where

$$C = \cos\theta \text{ and } S = \sin\theta.$$

Now, we can consider a pair of these equations as a system in four variables, (x, y, C, S), and proceed to solve (a1) and (b2), (b1) and (b2), and (a1) and (b1) for

x and y. For example, the intersection of two type (a) surfaces, (a1) and (a2) is a curve

$$p : r_{a1} \bigcap r_{a2} \rightarrow \Re^2 \times S^1$$

where $r_{a1} \bigcap r_{a2} \subset S^1$ denotes the intersected applicability constraints for (a1) and (a2). Although the solutions are in the variables C and S, we can use $C = \cos\theta = \cos r$ and $S = \sin\theta = \sin r$ to generate the curve of intersection in $\Re^2 \times S^1$.

After much simplification, the parametric solutions for the intersection curves for type (a) and (b) constraints in $\Re^2 \times S^1$ are as follows:

The intersection of two type (a) Surfaces: $(a1) \bigcap (a2)$

$$\begin{aligned}
p_\theta(r) &= r, \\
p_x(r) &= D\Big(-(S(\|a_i'\|\cos(\eta_i' + \lambda_i - \lambda_i') + \|a_i'\|\cos(\eta_i' - \lambda_i - \lambda_i') \\
&\quad - \|a_i\|\cos(\lambda_i' + \eta_i - \lambda_i) - \|a_i\|\cos(-\lambda_i' + \eta_i - \lambda_i) \\
&\quad + C(2\cos(\lambda_i' + \lambda_i - \gamma_j')\|b_j'\| - 2\cos(\lambda_i' + \lambda_i - \gamma_j)\|b_j\|)) \\
&\quad + C(\|a_i'\|\sin(\eta_i' + \lambda_i - \lambda_i') - \|a_i'\|\sin(\eta_i' - \lambda_i - \lambda_i') \\
&\quad - \|a_i\|\sin(\lambda_i' + \eta_i - \lambda_i) + \|a_i\|\sin(-\lambda_i' + \eta_i - \lambda_i)) \\
&\quad + C^2(2\sin(\lambda_i' + \lambda_i - \gamma_j')\|b_j'\| - 2\sin(\lambda_i' + \lambda_i - \gamma_j)\|b_j\|) \\
&\quad + (-\sin(\lambda_i' + \lambda_i - \gamma_j') - \sin(\lambda_i' - \lambda_i - \gamma_j'))\|b_j'\| \\
&\quad + (\sin(\lambda_i' + \lambda_i - \gamma_j) - \sin(\lambda_i' - \lambda_i + \gamma_j))\|b_j\|)/2\Big), \\
p_y(r) &= D\Big(-(S(\|a_i'\|\sin(\eta_i' + \lambda_i - \lambda_i') - \|a_i'\|\sin(\eta_i' - \lambda_i - \lambda_i') \\
&\quad - \|a_i\|\sin(\lambda_i' + \eta_i - \lambda_i) + \|a_i\|\sin(-\lambda_i' + \eta_i - \lambda_i) \\
&\quad + C(2\sin(\lambda_i' + \lambda_i - \gamma_j')\|b_j'\| - 2\sin(\lambda_i' + \lambda_i - \gamma_j)\|b_j\|)) \\
&\quad + C(-\|a_i'\|\cos(\eta_i' + \lambda_i - \lambda_i') - \|a_i'\|\cos(\eta_i' - \lambda_i - \lambda_i') \\
&\quad + \|a_i\|\cos(\lambda_i' + \eta_i - \lambda_i) + \|a_i\|\cos(-\lambda_i' + \eta_i - \lambda_i)) \\
&\quad + C^2(2\cos(\lambda_i' + \lambda_i - \gamma_j)\|b_j\| - 2\cos(\lambda_i' + \lambda_i - \gamma_j')\|b_j'\|) \\
&\quad + (\cos(\lambda_i' + \lambda_i - \gamma_j') - \cos(\lambda_i' - \lambda_i - \gamma_j'))\|b_j'\| \\
&\quad + (\cos(\lambda_i' - \lambda_i + \gamma_j) - \cos(\lambda_i' + \lambda_i - \gamma_j))\|b_j\|)/2\Big),
\end{aligned}$$

where

$$D = \csc(\lambda_i' - \lambda_i).$$

The Intersection of type (a) and (b) Surfaces: $(a1) \bigcap (b2)$

$$
\begin{aligned}
p_0(r) &= r, \\
p_x(r) &= F\Big(S(2C\|a_i'\| \cos(\phi_j' - \eta_i' - \lambda_i) \\
&\quad + \|b_j'\| \cos(\phi_j' + \lambda_i - \gamma_j') - \|b_j\| \cos(\phi_j' + \lambda_i - \gamma_j) \\
&\quad + \|b_j'\| \cos(\phi_j' - \lambda_i - \gamma_j') + \|b_j\| \cos(\phi_j' - \lambda_i + \gamma_j)) \\
&\quad + \|a_i'\| \sin(\phi_j' - \eta_i' + \lambda_i) - 2C^2 \|a_i'\| \sin(\phi_j' - \eta_i' - \lambda_i) \\
&\quad + \|a_i'\| \sin(\phi_j' - \eta_i' - \lambda_i) - \|a_i\| \sin(\phi_j' + \eta_i - \lambda_i) \\
&\quad - \|a_i\| \sin(\phi_j' - \eta_i + \lambda_i) \\
&\quad + C(\|b_j'\| \sin(\phi_j' + \lambda_i - \gamma_j') - \|b_j\| \sin(\phi_j' + \lambda_i - \gamma_j) \\
&\quad - \|b_j'\| \sin(\phi_j' - \lambda_i - \gamma_j') - \|b_j\| \sin(\phi_j' - \lambda_i + \gamma_j)) \Big), \\
p_y(r) &= -F\Big(S(2C\|a_i'\| \sin(\phi_j' - \eta_i' - \lambda_i) \\
&\quad - \|b_j'\| \sin(\phi_j' + \lambda_i - \gamma_j') + \|b_j\| \sin(\phi_j' + \lambda_i - \gamma_j) \\
&\quad + \|b_j'\| \sin(\phi_j' - \lambda_i - \gamma_j') - \|b_j\| \sin(\phi_j' - \lambda_i + \gamma_j)) \\
&\quad + \|a_i'\| \cos(\phi_j' - \eta_i' + \lambda_i) + 2C^2 \|a_i'\| \cos(\phi_j' - \eta_i' - \lambda_i) \\
&\quad - \|a_i'\| \cos(\phi_j' - \eta_i' - \lambda_i) - \|a_i\| \cos(\phi_j' + \eta_i - \lambda_i) \\
&\quad - \|a_i\| \cos(\phi_j' - \eta_i + \lambda_i) \\
&\quad + C(\|b_j'\| \cos(\phi_j' + \lambda_i - \gamma_j') - \|b_j\| \cos(\phi_j' + \lambda_i - \gamma_j) \\
&\quad + \|b_j'\| \cos(\phi_j' - \lambda_i - \gamma_j') - \|b_j\| \cos(\phi_j' - \lambda_i + \gamma_j)) \Big),
\end{aligned}
$$

where

$$
\begin{aligned}
F &= \frac{1}{(2\cos(\phi_j' - \lambda_i)S - 2C\sin(\phi_j' - \lambda_i))} \\
&= \frac{\csc(\theta - \phi_j' + \lambda_i)}{2}.
\end{aligned}
$$

The Intersection of two type (b) Surfaces: $(b1) \bigcap (b2)$

$$
\begin{aligned}
p_0(r) &= r, \\
p_x(r) &= E\Big((S(\|a_i'\|\cos(\phi_j' + \phi_j - \eta_i') - \|a_i\|\cos(\phi_j' + \phi_j - \eta_i) \\
&\quad - \|a_i'\|\cos(\phi_j' - \phi_j - \eta_i') + \|a_i\|\cos(\phi_j' - \phi_j + \eta_i)) \\
&\quad + C(-\|a_i'\|\sin(\phi_j' + \phi_j - \eta_i') + \|a_i\|\sin(\phi_j' + \phi_j - \eta_i) \\
&\quad + \|a_i'\|\sin(\phi_j' - \phi_j - \eta_i') + \|a_i\|\sin(\phi_j' - \phi_j + \eta_i)) \\
&\quad - \|b_j'\|\sin(\phi_j' + \phi_j - \gamma_j') + \|b_j\|\sin(\phi_j' + \phi_j - \gamma_j) \\
&\quad + \|b_j'\|\sin(\phi_j' - \phi_j - \gamma_j') + \|b_j\|\sin(\phi_j' - \phi_j + \gamma_j))/2\Big), \\
p_y(r) &= E\Big((S(\|a_i'\|\sin(\phi_j' + \phi_j - \eta_i') - \|a_i\|\sin(\phi_j' + \phi_j - \eta_i) \\
&\quad + \|a_i'\|\sin(\phi_j' - \phi_j - \eta_i') + \|a_i\|\sin(\phi_j' - \phi_j + \eta_i)) \\
&\quad + C(\|a_i'\|\cos(\phi_j' + \phi_j - \eta_i') - \|a_i\|\cos(\phi_j' + \phi_j - \eta_i) \\
&\quad + \|a_i'\|\cos(\phi_j' - \phi_j - \eta_i') - \|a_i\|\cos(\phi_j' - \phi_j + \eta_i)) \\
&\quad + \|b_j'\|\cos(\phi_j' + \phi_j - \gamma_j') - \|b_j\|\cos(\phi_j' + \phi_j - \gamma_j) \\
&\quad + \|b_j'\|\cos(\phi_j' - \phi_j - \gamma_j') - \|b_j\|\cos(\phi_j' - \phi_j + \gamma_j))/2\Big),
\end{aligned}
$$

where

$$E = \csc(\phi_j' - \phi_j).$$

I.2. Related Problems in $\Re^2 \times S^1$

I.2.1. Techniques for Moving Along C-Surfaces in $\Re^2 \times S^1$

In this section we present techniques for moving along a C-Surface. We could imagine using these methods to move to the nearest "edge" (C-Surface intersection), for example. A *level C-Surface* is defined *via* a function $f(x, y, \theta) = k$ for k constant. f is exactly of form $(a1)$ or $(b1)$ (above), and the level surface in $\Re^2 \times S^1$ is all points

$$L = \{ X \in \Re^2 \times r_f \mid f(X) = k \},$$

where $r_f \subset S^1$ is the θ applicability range for f.

Define a hyperplane in $\Re^2 \times S^1$ as the set

$$P = \{ X \in \Re^2 \times S^1 \mid X \cdot H = -h_4 \},$$

where $H = (h_1, h_2, h_3)$.

We intersect the level surface L with the hyperplane P to obtain the intersection curve:

$$\begin{aligned}
p_\theta(t) &= r, \\
p_x(t) &= G_1\Big(S(\cos(\lambda_i)(-h_3 t - h_4) + h_2 \sin(\lambda_i - \gamma_j)\|b_j\|) \\
&\quad + C \sin(\lambda_i)(-h_3 t - h_4) - h_2 \|a_i\| \cos(\lambda_i - \eta_i) \\
&\quad - C h_2 \cos(\lambda_i - \gamma_j)\|b_j\| - h_2 k \Big), \\
p_y(t) &= -G_1\Big(S(\sin(\lambda_i)(h_3 t + h_4) + h_1 \sin(\lambda_i - \gamma_j)\|b_j\|) \\
&\quad + C \cos(\lambda_i)(-h_3 t - h_4) - h_1 \|a_i\| \cos(\lambda_i - \eta_i) \\
&\quad - C h_1 \cos(\lambda_i - \gamma_j)\|b_j\| - h_1 k \Big), \qquad (P \cap (a1))
\end{aligned}$$

where

$$G_1 = \frac{1}{(h_2 \sin(\lambda_i) + h_1 \cos(\lambda_i))S + (h_1 \sin(\lambda_i) - h_2 \cos(\lambda_i))C}.$$

$$
\begin{aligned}
p_\theta(t) &= r, \\
p_x(t) &= -G_2\Big(\sin(\phi_j)(h_3 t + h_4) + h_2\|a_i\|\sin(\phi_j - \eta_i)S \\
&\quad + Ch_2\|a_i\|\cos(\phi_j - \eta_i) \\
&\quad + h_2\|b_j\|\cos(\phi_j - \gamma_j) + h_2 k\Big), \\
p_y(t) &= G_2\Big(\cos(\phi_j)(h_3 t + h_4) + h_1\|a_i\|\sin(\phi_j - \eta_i)S \\
&\quad + Ch_1\|a_i\|\cos(\phi_j - \eta_i) \\
&\quad + h_1\|b_j\|\cos(\phi_j - \gamma_j) + h_1 k\Big), \qquad (P \cap (b1))
\end{aligned}
$$

where

$$G_2 = \frac{1}{h_1\sin(\phi_j) - h_2\cos(\phi_j)}.$$

I.2.2. Characterizing Clearance to a C-Surface

It would be very useful to characterize the minimum clearance to a C-surface. We would like to answer the question:

- *For a point $b_{xy} \in \Re^2$, at what orientation is b_{xy} closest to a C-surface, and what is minimum directed clearance vector at that orientation?*

Using Lagrange multipliers, we can minimize a function $f(x, y, \theta)$ subject to a constraint $g(x, y, \theta) = 0$ by constructing the auxiliary function

$$H(x, y, \theta, \ell) = f(x, y, \theta) - \ell g(x, y, \theta)$$

and solving the partial derivatives

$$\frac{\partial H}{\partial x} = 0 \qquad (1)$$

$$\frac{\partial H}{\partial y} = 0 \qquad (2)$$

$$\frac{\partial H}{\partial \theta} = 0 \qquad (3)$$

$$\frac{\partial H}{\partial \ell} = 0. \qquad (4)$$

In our case, g will define a C-surface, for example, a type (a) surface:

$$g(x, y, \theta) = \sin(\theta + \lambda_i)y + \cos(\theta + \lambda_i)x - \|b_j\|\cos(\theta + \lambda_i - \gamma_j) - \|a_i\|\cos(\lambda_i - \eta_i)$$

and f will be a distance function. Now, the rotational dimensions cannot be treated uniformly in establishing a metric, so we will define distance in Euclidean space. Minimizing the square of the translational distance suffices for our purposes. Hence,

$$f(x, y, \theta) = (x - b_x)^2 + (y - b_y)^2.$$

Differentiating H gives us the following equations:

$$\frac{\partial H}{\partial x} = 2(x - b_x) - \ell \cos(\theta + \lambda_i) \tag{1}$$

$$\frac{\partial H}{\partial y} = 2(y - b_y) - \ell \sin(\theta + \lambda_i) \tag{2}$$

$$\frac{\partial H}{\partial \theta} = -\ell(\cos(\theta + \lambda_i)y - \sin(\theta + \lambda_i)x + \|b_j\| \sin(\theta + \lambda_i - \gamma_j)) \tag{3}$$

$$\frac{\partial H}{\partial \ell} = (-\sin(\theta + \lambda_i)y - \cos(\theta + \lambda_i)x + \|b_j\| \cos(\theta + \lambda_i - \gamma_j) + \|a_i\| \cos(\eta_i - \lambda_i)) \tag{4}$$

Solving these equations for x, y, θ, and ℓ is not trivial. However, the following observations make it easier. First of all, we note that solving

$$\ell \frac{\partial H}{\partial \ell} = 0 \tag{5}$$

is equivalent to solving (4) as long as $\ell = 0$ is not a solution. We next solve (1) and (2) for $\ell \cos(\theta + \lambda_i)$ and $\ell \sin(\theta + \lambda_i)$ and substitute this value in to (3) and (5). (5) then becomes a linear equation in x and y while (3) is quadratic in x and y and linear in ℓ. Our rewriting of (4) into (5) has thus eliminated ℓ from (5), and we can solve for x in terms of y:

$$-2y^2 + (2\|b_j\| \sin(\gamma_j) + 2b_y)y - 2x^2 + (2\|b_j\| \cos(\gamma_j) + 2b_x)x + \ell\|a_i\| \cos(\eta_i - \lambda_i) - 2b_y\|b_j\| \sin(\gamma_j) - 2b_x\|b_j\| \cos(\gamma_j) \tag{3}$$

$$(2b_x - 2\|b_j\| \cos(\gamma_j))y + (2\|b_j\| \sin(\gamma_j) - 2b_y)x - 2b_x\|b_j\| \sin(\gamma_j) + 2b_y\|b_j\| \cos(\gamma_j) \tag{5}$$

We need one additional constraint: this is obtained by observing that

$$\sin(\theta + \lambda_i)^2 + \cos(\theta + \lambda_i)^2 = 1. \tag{6}$$

Since the trigonometric terms can be expressed in x, y, and ℓ, we can obtain ℓ^2 in terms of x and y. (3), (5), and (6) then result in a quartic in x with the roots:

$$x = G_1\Big(2b_y\|b_j\|^2\sin(2\gamma_j) + 2b_x\|b_j\|^2\cos(2\gamma_j)$$
$$- ((2b_y^2 + 2b_x^2)\|b_j\| + 2\|b_j\|^3)\cos(\gamma_j) + 2b_x\|b_j\|^2$$
$$\pm G_2\Big(\|a_i\|\|b_j\|\cos(\eta_i - \lambda_i + \gamma_j)$$
$$+ \|a_i\|\|b_j\|\cos(\eta_i - \lambda_i - \gamma_j) - 2b_x\|a_i\|\cos(\eta_i - \lambda_i)\Big)\Big),$$

where

$$G_1 = \frac{1}{2\Big(2b_y\|b_j\|\sin(\gamma_j) + 2b_x\|b_j\|\cos(\gamma_j) - \|b_j\|^2 - b_y^2 - b_x^2\Big)}$$

and

$$G_2 = \sqrt{-2b_y\|b_j\|\sin(\gamma_j) - 2b_x\|b_j\|\cos(\gamma_j) + \|b_j\|^2 + b_y^2 + b_x^2}.$$

Given x, y is found from (5) :

$$y = \frac{(\|b_j\|\sin(\gamma_j) - b_y)x - b_x\|b_j\|\sin(\gamma_j) + b_y\|b_j\|\cos(\gamma_j)}{\|b_j\|\cos(\gamma_j) - b_x}.$$

ℓ can be found from (3) as a linear function of x and y. To determine θ, we calculate $\sin(\theta + \lambda_i)$ and $\cos(\theta + \lambda_i)$ from

$$\sin(\theta + \lambda_i) = \frac{2(y - b_y)}{\ell}$$
$$\cos(\theta + \lambda_i) = \frac{2(x - b_x)}{\ell}$$

and use a two argument arctangent function $Atan2 : \Re^2 \rightarrow S^1$ to determine $\theta + \lambda_i$. The θ value must be checked against the applicability constraints for surface g; if it falls outside the range, then endpoints will yield the minimum clearance. Naturally, it is possible that for certain orientations, (b_x, b_y, θ) will lie on or inside the C-surface. These cases may be disambiguated by the sign of $g(b_x, b_y, \theta)$. Finally, given the closest point (at some orientation θ) on the C-Surface, the minimum clearance is simply the vector

$$(x, y) - (b_x, b_y).$$

The Minimum Clearance to a type (b) C-surface

To find the minimum clearance to a type (b) surface and the orientation at which the clearance occurs, we let g be a type (b) constraint (equation (b1)) and solve the system of partial derivatives of H.

$$\frac{\partial H}{\partial x} = 2(x - b_x) - \ell\cos(\phi_j) \tag{1}$$

$$\frac{\partial H}{\partial y} = 2(y - b_y) - \ell\sin(\phi_j) \tag{2}$$

$$\frac{\partial H}{\partial \theta} = -\ell\|a_i\|\sin(\theta - \phi_j + \eta_i) \tag{3}$$

$$\frac{\partial H}{\partial \ell} = -g(x, y, \theta) \tag{4}$$

The solution is considerably easier because the form of the constraint surface is less complicated. Since $\partial H/\partial \ell = -g$, θ may be found in terms of x and y using an arccosine. Substituting this value of θ into (3) yields a quadratic equation in x, y, and ℓ, which when solved with (1) and (2) for the following roots:

$$\begin{aligned}
x &= \|b_j\|\cos(\phi_j)\cos(\phi_j - \gamma_j) \\
&\quad - b_y\cos(\phi_j)\sin(\phi_j) - b_x\cos(\phi_j)^2 - \|a_i\|\cos(\phi_j) + b_x, \\
y &= \|b_j\|\sin(\phi_j)\cos(\phi_j - \gamma_j) \\
&\quad - (b_x\cos(\phi_j) \pm \|a_i\|)\sin(\phi_j) + b_y\cos(\phi_j)^2, \\
\ell &= 2\|b_j\|\cos(\phi_j - \gamma_j) - 2b_y\sin(\phi_j) - 2b_x\cos(\phi_j) - 2\|a_i\|.
\end{aligned}$$

Appendix II

The Connectivity of Configuration Space

II.1. A Review of Elementary Homotopy Theory

In this appendix we review some elementary homotopy theory, and address the connectivity of configuration space. See Hocking and Young (1961) for a more extensive review, and Donald (1983a) for an analysis of the relation between channels and homotopic equivalence classes. Let I^1 denote the unit interval. A parameterized family of mappings from a space X into a space Y is a continuous function $h : X \times I^1 \to Y$. Consider the mappings f and g from X to Y: we say that h is a *homotopy* between f and g if for each point x in X,

$$h(x,0) = f(x) \text{ and } h(x,1) = g(x).$$

Intuitively the existence of h implies that f can be continuously deformed into g without leaving Y.

The homotopy relation between mappings from X into Y is an equivalence relation on the function space Y^X. Hence the homotopy relation partitions Y^X into disjoint equivalence classes, which are called *homotopy classes*. We write the homotopy relation as $f \simeq g$. These homotopy classes capture our intuitive notion of classes of paths. The homotopy classes of Y^X can be shown to be precisely the arcwise-connected components of Y^X (Hocking and Young (1961)).

To take a concrete example, consider configuration space for the two-dimensional mover's problem to be the product space of the 2-dimensional Euclidean plane $\Re^2$ and the one-dimensional sphere S^1 to obtain $\Re^2 \times S^1$, and denote the configuration obstacles as $CO \subseteq \Re^2 \times S^1$. Now two paths f and g in the same equivalence class *must* belong to a parameterized family of mappings such that:

$$h : X \times I^1 \rightarrow \Re^2 \times S^1 - CO.$$

and $h(x,0) = f(x)$, $h(x,1) = g(x)$ as before.

Now, let y be a point in Y. The y neighborhood of cyclic paths in Y, $C(Y,y)$ is the collection of all continuous mappings $f : I^1 \rightarrow Y$ such that $f(0) = f(1) = y$, i.e., the set of all continuous curves that begin and end at y. If f and g are curves in $C(Y,y)$, we say that f is *homotopic to g modulo y* if there exists a homotopy $h : I^1 \times I^1 \rightarrow Y$ continuously deforming f into g without leaving Y.

Clearly, homotopy modulo y is an equivalence relation, and decomposes $C(Y,y)$ into disjoint equivalence classes which are exactly the arcwise-connected components of $C(Y,y)$. The set of these equivalence classes is termed the *first homotopy group*, or *fundamental group* of Y. We say a path-connected space Y is *simply-connected* if the fundamental homotopy group for X is the trivial group of one element (for some, and hence for all y in Y). See also appendix III, section "Topological Constraints."

II.2. The Connectivity of Configuration Space

The configuration space $\Re^2 \times S^1$ (for the two-dimensional mover's problem) is not simply-connected, since S^1 is not simply-connected. The function space $(\Re^n \times S^1)^X$ contains several homotopy classes. $(\Re^2 \times S^1)^X$ may be envisioned as a cylinder on which there are clearly two classes of paths: those that bound a 2-dimensional region and are contractable to a point, and those that go around the cylinder.

The configuration space $\Re^3 \times SO(3)$ is not simply connected, because $SO(3)$ is not simply connected. To see this consider the following: geometrically, $SO(3)$

is homeomorphic to P^3, the 3-sphere with antipodal points identified. As is well known (see Massey (1967), p. 166) the fundamental group for P^n is cyclic of order 2, and hence P^n is not simply connected.

General configuration spaces (other than that for the classical Mover's problem) are not always simply-connected. For example, the *C-Space* for a manipulator with six revolute joints is the 6-torus, $S^1 \times S^1 \times \ldots \times S^1$ (to 6).

Let Π be the half-open interval $[-\pi, \pi)$. Π can be used to approximate S^1, if we are willing to tolerate singularities in the representation. It is instructive to generate a configuration space which is simply-connected. Since this is not possible for the general product space $\Re^n \times S^1$ we will instead consider the product space of $\Re^n$ and Π. Thus for the two-dimensional mover's problem we consider the product space

$$C = \Re^2 \times \Pi.$$

For a manipulator with m revolute joints, the *C-Space* may be approximated by Π^m, where

$$\Pi^m = \Pi \times \Pi \times \ldots \times \Pi \qquad (\text{to } m).$$

The three dimensional rotation group $SO(3)$ can be approximated by a hemisphere of the 3-sphere (which is simply connected), or by Π^3. Π^m is homeomorphic to the interior of the m-cube. This new product space C is simply a restricted configuration space where the piano is not allowed to spin around wildly. The approximation of $SO(3)$ by a hemisphere of S^3, incidentally, is closely related to the employment of unit quaternions to represent rotations. The space of unit quaternions is precisely S^3; the two quaternions q and $-q$ construct the same rotation, although they represent antipodal points on S^3. When all antipodal points q and $-q$ are identified, the projective 3-sphere, P^3 is obtained. P^3 is isomorphic to $SO(3)$. It is of interest that Euler angle space, Q^3 (see chapter 2) is essentially an approximation of $SO(3)$ by $\Re^3$ (mod 2π), which is isomorphic to the 3-torus,

when equivalent rotations are identified. If equivalent rotations are not identified, then Q^3 is isomorphic to $\Re^3$. While the approximation of $SO(3)$ by $\Re^3$ yields a simply-connected configuration space, from the point of view of an automated planner it has the undesirable effect of introducing an infinite number of goals in the rotational dimensions of configuration space, for every single goal in the space of Euclidean motions. For this reason the approximation provided by the 3-torus may be considered preferable.

Appendix III

Integrating Local and Global Algorithms for the Find-Path Problem

In Donald (1983a), we have discussed the integration of a global channel algorithm with a local *C-space* algorithm to form a planning system for the find-path problem in $\Re^2 \times S^1$. How can a three dimensional global, or channel algorithm be coupled with the planning system in $\Re^3 \times SO(3)$ described in previous chapters? More generally, what are the fundamental issues in integrating local and global geometric planning algorithms? In particular,

(i) How can a global algorithm suggest paths, or equivalence classes of paths to a local algorithm?

(ii) How can the relevant geometric constraints be identified and exploited by the local algorithm? Conversely, how can irrelevant geometric constraints be effectively ignored?

(iii) How can global topological constraints, such as those arising from analysis of homotopy classes and fundamental groups, be propagated onto the (local) geometric structure examined by the local algorithm?

In general, the design of a global algorithm will depend on the geometric constraints exploited by the companion local algorithm with which it will be coupled. Hence when we consider extending the channel algorithm of Donald (1983a) to the three dimensional find-path domain, we must specify what "target," local algorithm to use. A natural candidate is the local algorithm for find-path in $\Re^3 \times SO(3)$, which we describe in chapters 1-2.

Path Suggestion. A problem which must be solved in any local/global find-path integration is how a global path may be suggested to a local algorithm. In two dimensions this was accomplished by segmenting the find-path problem into a sequence of sub-problems. The Suggestor strategy (chapter 2) is designed with this in mind. The verified points along the suggested path become planning islands in configuration space. The job of the local algorithm is then to connect up the planning islands and find a continuous path to the goal.

Choosing Subgoals in Rotation Space. In a three-dimensional rotation space, the problem of selecting good rotational subgoals becomes more difficult. Much of the path-planning literature has been guilty of overlooking this difficulty. Such subgoals can be used in path-suggestion as we have described above. Even when the companion local algorithm is complete, a strategy for choosing good rotational subgoals is desirable, since it would allow the algorithm to converge faster.

We have derived experimental strategies which consider *alignments* of the robot polyhedra with the faces and edges of obstacles. Every polyhedron's boundary contains *alignable* generators (faces and edges) which have an orientation, and *non-alignable* generators (vertices) which have no orientation. Two generators are said to be aligned when they are parallel, and the rotations in which they are aligned form connected *alignment regions* in $SO(3)$. For example, two cubes are aligned when two faces are parallel, or an edge and a face are parallel. The channel construction is useful for identifying the obstacle surfaces which bound the proposed channel (in real space). Call this set of faces $\mathcal{F}_K$. Let $\mathcal{F}_R$ denote the faces of the robot. The alignment regions can be considered for the generator pairs

$$G_{Align} = \Big(\mathcal{F}_R \times \mathcal{F}_K\Big) \bigcup \Big(\mathcal{F}_R \times \hat{\partial}\mathcal{F}_K\Big) \bigcup \Big(\hat{\partial}\mathcal{F}_R \times \mathcal{F}_K\Big).$$

We are only interested in *applicable* alignments, that is, an alignment of two (or more) generators where the generators can be brought into contact through some translational motion (this is our definition of applicability: see chapter 3). Applicability may be determined by examining the applicability constraint functions (ACFs) introduced in chapter 3. Furthermore, in chapter 5, we showed

that alignments of edges and faces occurred exactly at the boundary of the applicability regions for C-functions.

Every C-function is a partial function f_i on the configuration space $\Re^3 \times SO(3)$, whose domain is $\Re^3 \times A_i$ for $A_i \subset SO(3)$. The set of alignment regions is obtained by the union of the boundaries of these applicability regions

$$R_{align} = \bigcup_i \partial A_i$$

for every C-function f_i. Every point in R_{align} lies in the kernel of some ACF $g_{i,j} : SO(3) \rightarrow \Re$ for a C-function f_i. In chapter 4 we showed how to derive charts for these boundaries and their intersection manifolds.

In most find-path problems, the alignment regions in R_{align} are complete as subgoals in rotation space—i.e., no other rotational subgoals need be considered in order to find a solution path (if one exists). This makes a certain intuitive sense: one might try aligning a large box with a narrow door-frame in order to squeeze it through.[1] However, in general there exist pathological cases in which this is not true (imagine a robot which looked like a polyhedral sea urchin), and the set of alignments is not complete as a set of subgoals. Furthermore, it is unsatisfying that the alignment analysis exploits strong constraints in the polyhedral domain of the classical Movers' problem, and does not appear to generalize well to linked-arm problems.

We believe that it may be possible to overcome the problem of "star-shaped" robots by considering additional alignment regions obtained from faces and edges of the convex hull of such objects. Such an algorithm would have to deal similarly with "star-shaped" obstacles. Even these additional alignments may prove incomplete; however, they may have heuristic value.

The problem of how a global algorithm can infer good rotational subgoals from the structure of real-space is one of the most interesting open problems in spatial planning. We conjecture that an answer may lie in the structure of the boundaries

[1] We are not making *any* claims about human spatial reasoning here.

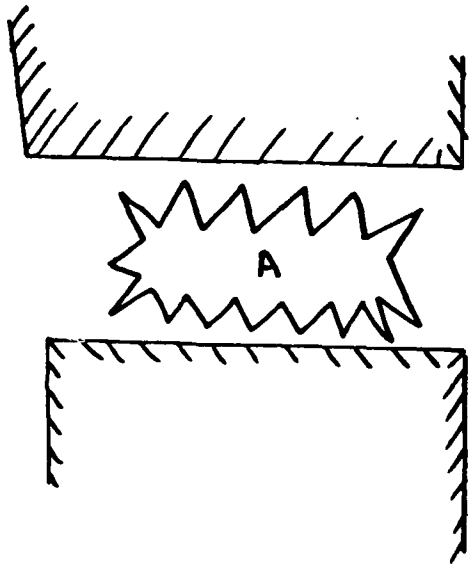


Figure II.1. Pathological example showing a robot A whose alignment regions do not include a rotation which helps get through the tight passage.

of the applicability regions in $SO(3)$. The algorithms we provide in chapter 5 for obtaining the applicability set decomposition may prove useful in computing this structure. Such a planner might slide along the intersections of ACF boundaries or *level* ACFs in rotation space,[2] much as our planner slides on the intersections of level C-surfaces in configuration space. The advantage of such an approach lies in reducing the (infinite) search for rotational subgoals to a finite combinatorial search along the ACF boundaries.

Topological Constraints. Ideally, the global strategy should enforce the path-class criterion[3] for each sub-problem: no straight-line approximation for a subproblem may cross more than one equivalence class of paths. We begin by defining what a straight-line approximation means in $\Re^3 \times SO(3)$. This requires some way of forming "linear combinations" of rotations. The definition of "linear combinations"

[2]Level ACFs are defined in section 3.11.
[3]See and appendix II.

in $SO(3)$ relates to a definition of convexity for $\Re^3 \times SO(3)$. In particular, we wish to determine whether *C-space* obstacles are convex. For if *C-space* obstacles are convex, then we could trivially bound the number of intersections any straight-line trajectory can make with any one obstacle. We present a conjecture that the *C-space* obstacles are non-convex. Finally, we discuss basic topological notions for formalizing our analysis of equivalence classes of paths.

We require some way of forming "linear combinations" of rotations. The requisite algebraic structure is much like a *module* (over the reals), except that the group operation cannot be commutative.[4]. The group operation is composition of rotations. Let $\mathcal{R}(\hat{n}, \theta)$ denote rotation about the three dimensional vector $\hat{n}$ by angle θ. Scalar multiplication by $\alpha \in \Re$ may be defined by

$$\alpha\mathcal{R}(\hat{n}, \theta) = \mathcal{R}(\hat{n}, \alpha\theta).$$

By substituting the (non-commutative) composition of rotations for the group operation $+$, we obtain a natural definition for linear combinations,

$$\alpha\mathcal{R}(\hat{n}, \theta) + (1-\alpha)\mathcal{R}(\hat{n}', \theta') = \mathcal{R}(\hat{n}, \alpha\theta)\mathcal{R}(\hat{n}', (1-\alpha)\theta') \qquad (III.1)$$

for $0 \leq \alpha \leq 1$. $\mathcal{R}$, of course, may be conveniently expressed by a unit quaternion.

Suppose Q^3 is a three-dimensional parameter space for $SO(3)$—that is, the domain of a chart for rotation space. For example, Q^3 might be the space of Euler angles (see chapter 2). It is possible to define linear combinations in the parameter space $\Re^3 \times Q^3$. This seems unsatisfactory, since it makes the definition of linear combination—and more disturbingly, of convexity—dependent upon the chosen parameterization for $SO(3)$. Observe that definition (III.1) for linear combinations is invariant for all parameterizations.

Open Question: Under a definition of convexity invariant for all parameterizations, show whether or not the *C-space* obstacles in $\Re^3 \times SO(3)$ (and $\Re^2 \times S^1$)

[4]Recall that a *module* is defined as follows: If R is a *commutative ring with identity*, then M is a *module* over R if $(M, +)$ is a commutative group, and scalar multiplication $(r, M) \mapsto rM$ of elements M in M by r in R is associative and distributive (over $+$), and if $1_R M = M$

are convex. *Conjecture:* We conjecture that *C-space* obstacles are non-convex. When $\Re^2 \times S^1$ is approximated by $\Re^2 \times [-\pi, \pi)$ and embedded in $\Re^3$, the corresponding *C-space* obstacles are non-convex using Euclidean linear combinations. Furthermore, in both $\Re^2 \times S^1$ and $\Re^3 \times SO(3)$, each obstacle manifold is the intersection of a finite number of half-hyperspaces of $\Re^3 \times SO(3)$. Each half-hyperspace is in turn defined via a real-valued partial function $f : \Re^3 \times SO(3) \rightarrow \Re$. Using partial functions, arbitrary non-convex manifolds can be constructed. Showing that the obstacle manifolds could be represented by means of smooth, total functions would suggest convexity. Our analysis suggests that these functions must be partial, which in turn leads to the conjecture that the obstacle manifolds are non-convex. ∎

The homotopy relation (see appendix II) partitions the function space of paths into equivalence classes. The *image* of one such equivalence class $[f]$ is the region in *C-space* covered by the union of all the path images in $[f]$. The equivalence classes are determined both by the structure of the underlying *C-space* , and by the *C-space* obstacles. Intuitively, the fundamental group[5] in a space Y is a topological invariant corresponding to the set of equivalence classes of paths in Y. The group operation corresponds to path composition ("pasting"), and for two paths $f, g : I^1 \rightarrow Y$ where $f(1) = g(0)$,

$$f * g\,(t) = \begin{cases} f(2t), & \text{for } t \in [0, \frac{1}{2}] \\ g(2t-1), & \text{for } t \in [\frac{1}{2}, 1]. \end{cases}$$

We think of $f * g$ as the path whose first half is f and whose second half is g. The pasting operation $*$ is well defined on path homotopy classes:

$$[f] * [g] = [f * g],$$

and exhibits groupoid properties. When an obstacle makes a hole in free-space, it augments the fundamental group for the space by adding an infinite (cyclic) number

[5]See appendix II for a review of elementary homotopy theory and a formal definition of the fundamental group.

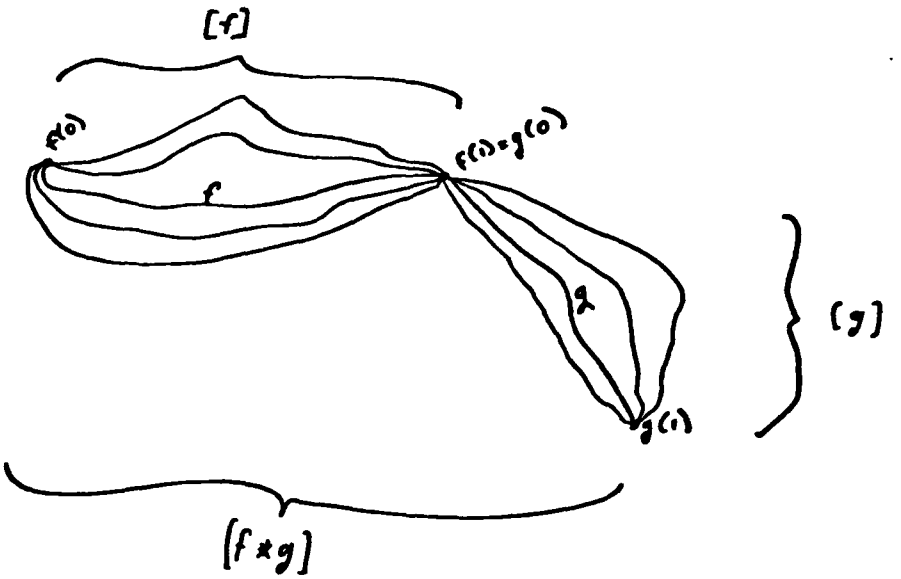


Figure II.2. Pasting together of paths f and g. Some paths in the homotopic equivalence classes $[f]$ and $[g]$ are also shown.

of equivalence classes. (For example, by puncturing the plane at the origin O, we obtain the classes of paths (1) not looping around O, (2) looping around O once, ..., (n) looping $n-1$ times around O, ...). The topology of the underlying *C-space* may be predetermined (see appendix II), but each new find-path environment generates different path homotopy classes. We wish to infer the equivalence classes in the fundamental group by their *generators*, i.e., the *C-space* obstacles. Since the *C-space* obstacles can be constructed from the real-space obstacles, we are actually attempting to compute path classes in *C-space* from the structure of real-space.

In general, if free-space is connected, the image in *C-space* of even a single class of paths can cover all of free-space. However, we can impose a stronger condition which subsumes the path-class criterion. Let p be an injection of I^1 into *C-space* , which will represent some approximation of a solution path for a subproblem Ξ. We wish to know whether the image of p can be expressed as the union of two

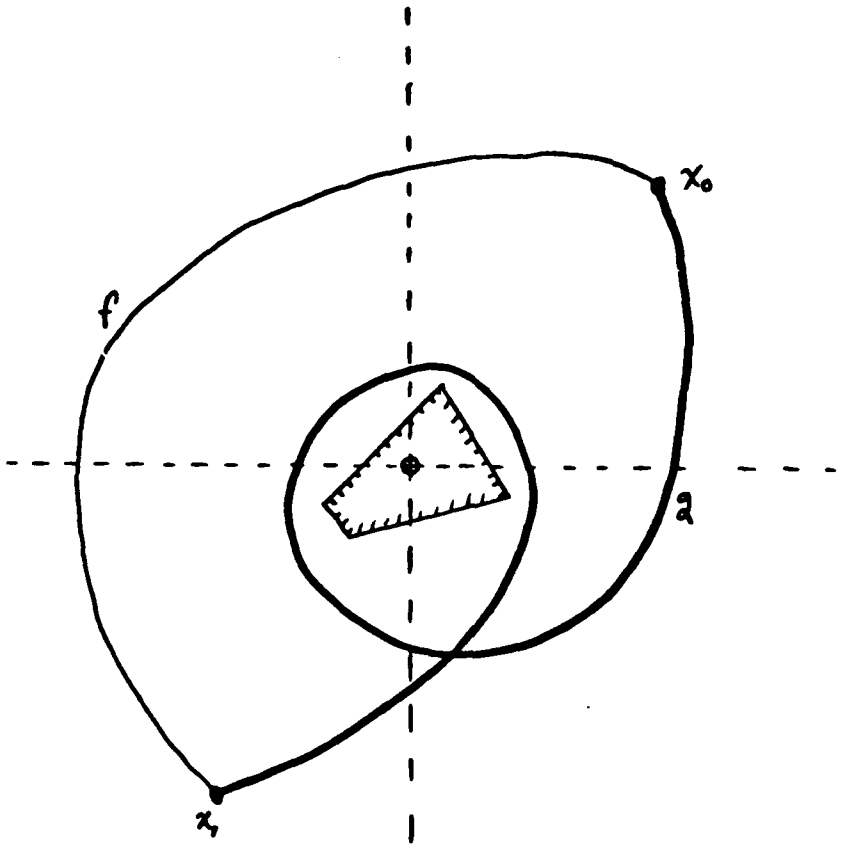


Figure II.3. The plane with a puncture (or obstacle) at the origin, showing paths f and g from x_0 to x_1. f and g are not homotopically equivalent, and hence in different equivalence classes.

sets of points, those lying in the image of one equivalence class of paths, and those lying in *C-space* obstacles bounding that image. In chapter 4, we showed how to intersect trajectories with C-surfaces in $\Re^2 \times S^1$ and $\Re^3 \times SO(3)$. We can intersect $p(I^1)$ with all C-surfaces, and determine the intersection points. These intersection points indicate where it penetrates *C-space* obstacles, and are determined from its zero-crossings from free-space to forbidden space.

Claim: That the image of p contains either no region or one connected region lying within any *C-space* obstacles is a sufficient (but not a necessary) condition satisfying the path-class criterion for sub-problems.

Constraint Relevance. Another issue concerns how a global algorithm can characterize the relevant constraints for a local algorithm, and if necessary, impose additional, artificial constraints on the problem so that irrelevant constraints in the initial domain will be ignored. Because of the difficulties in maximizing channel

breadth in three dimensional channel construction, the artificial faces introduced by the current implementation may prove too restrictive, especially if the robot is large or non-convex. However, the channel construction is useful for identifying the obstacle surfaces which bound the proposed channel (in real space). Call this set of faces $\mathcal{F}_K$. The C-surfaces generated by these faces may be exploited by the *sliding* and *intersection* experts (see chapter 2). Let $\mathcal{F}_R$ denote the faces of the robot. These preferred C-surfaces are identified with their generator pairs, namely

$$Slide = \Big(\mathcal{F}_R \times vert(\mathcal{F}_K)\Big) \bigcup \Big(\hat{\partial}\mathcal{F}_R \times \hat{\partial}\mathcal{F}_K\Big) \bigcup \Big(vert(\mathcal{F}_R) \times \mathcal{F}_K\Big).$$

The identification of good C-surfaces to slide along addresses a central issue in local/global integration. At present, the sliding and intersection experts exploit only local geometric structure and planning history. The channel algorithm introduces a global criterion for selecting which C-surfaces to slide along, and for constructing intersection manifolds. The set of C-surfaces *Slide* specifies an implicit volume in configuration space which is closely related to the channel volume (in *C-space*). This volume is obtained by extending the hyperplanes containing the C-surfaces past the obstacle boundaries until they intersect. Furthermore, *Slide* lies on the boundary of the image of the hypothesized equivalence class of paths. By choosing these interior surfaces as candidates for the sliding and intersection experts, global advice on constraint relevance is provided to the local *C-space* algorithm by the global channel algorithm.

Appendix IV

A Listing of Macsyma Code

In this appendix, we provide a listing of the Macsyma code to produce optimized Lisp procedures for computing the coefficient functions of the canonical linear forms and trigonometric quadratic forms of the type (a), (b), and (c) C-functions, and type (a) and (b) ACF clauses for $\Re^3 \times SO(3)$. Using these forms, the intersection manifolds, type (c) ACFs, and disambiguating applicability constraints are constructed in the manner described in the text.

We also list (in Macsyma) the resulting combined forms for the C-functions and ACFs. Note that the type (c) C-function is "over a page long."

Rotations are specified by means of the Macsyma functions $RotateVector(x : vector)$ and $Transform(x : plane \quad or \quad vector)$. Rotations are implemented using the Euler angles. However, by changing these two functions, any representation for rotations such as quaternions, spherical angles, or wrist angles for a cartesian manipulator may be employed. This corresponds to reparameterizing $SO(3)$, and results in different charts for the level C-surfaces, intersection manifolds, and ACF manifolds.

```
/* Display and Grind function. If OPTIMIZE_FLAG is TRUE then
   we actually store the OPTIMIZED expression */

print("Setting Grind to TRUE...")$
Grind:True$
Optimize_Flag: TRUE$
/*   OPIMPREFIX: %    */

Display_and_Grind(exp):=
Block([label],
             if OPTIMIZE_FLAG THEN
               (print(EXP),
                     Print(" Optimizing..."),
                     Exp: Optimize(Exp)),
       label:Ldisp(exp),
       if Grind then
         (print("                Ground, becomes:
", label[1]," : "),
                grind(exp)),
        print(" "),
        label[1])$


/* Utility function. Is the expression EXP free of all the VARS (a list?) */

Free_Of_Vars(Vars, Exp) :=
block([freedom],
             freedom:true,
        (for var in Vars  unless freedom = false do
            (if Not FreeOf(Var, Exp) then freedom: false)),
         if Not freedom then
Print("[Exp contains Major Variables. Recursively Analyze...]"),
          freedom)$

/* here we define Canonical Linear Form to be simply the
   expression of the constraint as a linear function
   in X, Y, and Z */

Canonical_linear_variables: [X, Y, Z]$

Canonical_linear_form(Exp) :=
        IsolateN(Exp, Canonical_linear_variables)$
```

```
/* Bruce Donald (BRD@OZ) analyze hairy expressions -*- MACSYMA -*-
   a little bit */
/*  Analyze BiLinear Forms : given the "chief vars" in RATVARS,
    generate intermediate labels for all the coefficients
    of these vars and return the "simplified" bilinear form.
    Recursively Calls ANALYZE_BILINEAR_FORM So that the intermediate
    labels are truly "constants" relative to the RatVars */

/* typically, ratvars:[x,y,z,psi,theta,phi] */

/*  IsolateN works like ISOLATE but for N variables in a list,
    on a bilinear form */

IsolateN(Exp, Nvars) :=
Block([Save_Ratvars, Iform],
                  Save_ratVars: ratvars,
                  RatVars: Nvars,
                  Iform: Analyze_Bilinear_form(Exp),
                  Ratvars: Save_Ratvars,
                  Iform)$

simple_Analyze_depth: 4$

Analyze_Bilinear_Form(exp) :=
   block([power, Coef, Rat_Exp,  Left,  lose, Sum, Label],
         print("Analyzing:"),
         ldisp(reveal(Exp,simple_analyze_depth)),
   Left: rat(exp), Sum: 0,
      for var in Ratvars do
        (Power: Hipow(Left, Var),
             if power > 2 then
             (lose: var^power,
                    Error("Warning: Not a Bilinear Form because of ", lose)),
          Coef: ratcoeff(Left, var, 2),
         if not (Coef = 0) then
         ( Print( "The coefficient of ", var^2," is "),
              Label:Display_and_grind(Coef),
              if not Free_of_vars(ratvars,Coef)
                  then Label: Analyze_Bilinear_form(Coef),
              Sum: Sum + label * Var^2,
              Left:  rat(left - Coef * (Var ^2)))),
   print("Mixed terms: "),
   for var in Ratvars do
    (for var2 in Ratvars do
      if var #var2 then
        (Coef: ratcoeff(Left, var*var2, 1),
         if not (Coef = 0) then
            ( Print( "The coefficient of ", Var*var2," is "),
                 label:Display_and_grind(Coef),
                 if Not Free_Of_vars(RatVars, Coef)
                     then Label: Analyze_Bilinear_form(Coef),
                 Sum: Sum + label * var* var2,
                 Left:  rat(left - Coef * (Var * var2))))),
   print(" Linear Terms: "),
   for var in Ratvars do
     (coef: ratcoeff(left, var, 1),
           if not (coef = 0) then
           (  print("The Coefficient of ",var," is "),
                label:Display_and_grind(Coef),
                if Not Free_Of_vars(RatVars, Coef)
                    then Label: Analyze_Bilinear_form(Coef),
                Sum: Sum + label*var,
                Left: rat(left - Coef* Var))),
   if Left # 0 then
      (Print(" And the constant term is "),
           Label: Display_and_grind(left),
           sum: Sum + Label),
   Print(" Yielding :"), Display_and_grind(Sum), Sum)$
```

```
/* Bruce R. Donald. (BRD@OZ)    -*- Mode:Macsyma -*-
  Attempt to express applicability constraints for
  C-surfaces in R^3 \cross S^3 */

/* PRODUCTION VERSION -- i.e., for production of LISP code */

BTHETA: [phi, theta, psi]$

if .Euler_Rotation_equations_loaded = TRUE then "OK"
    else Batchload ([rotate.mac]);

shorten(exp):= subst( S,sin, subst(C, cos, exp))$

ratvars:[ sin(phi),cos(phi),sin(theta),cos(theta), sin(psi),cos(psi)];

/* Each constraint is of the form */

  /* vectors: */
  u(i) := [ux[i],uy[i],uz[i]];
  v(i) := [vx[i],vy[i],vz[i]];

  /* Normal for plane eq */

  n(i) := [nx[i],ny[i],nz[i],nd[i]];

/* Here we define functions to generate the applicability constraints.
   the arguments are: Bn : a vertex in 3-space, which we use to
                           measure distance to the plane.
                      v  : a vertex which we insist must be ON the plane.
                      N  : a plane (4-vector)
                      c  : the "height" of the level surface in S^3.
                           if 0, corresponds to the maximim boundary
                           of the applicability clause (eg edge-face
                           contact).
*/


R3_projection(Vec) := [vec[1], vec[2], vec[3]]$

Type_B_Clause(N, bn, v, c) :=
                          simp_3(
                            (R3_Projection(N) . Rotate_vector(Bn))
                          - (R3_Projection(N) . Rotate_vector(v))
                          - c);

Type_A_clause(N, bn, v, c) :=
                    Block([N_THETA],
                     N_THETA: part(transform(Euler_Inverse, N), 1),
                     simp_3(
                       (R3_Projection(N_THETA) . bn)
                     -  (R3_Projection(N_theta) . v)
                     -c));

Midp(a,b) := a +  (b-a)/2$

Type_C1_Clause(n1, n2, a1, a2, c1, c2) :=
                    simp_3(
                    - type_b_clause(n1, a1, midp(a1,a2), c1)
                    * type_b_clause(n2, a1, midp(a1,a2), c2));

Type_C2_Clause(N1,N2, b1, b2, c1, c2) :=
                    simp_3(
                     - type_a_clause(n1, b1, midp(b1,b2), c1)
                     * type_a_clause(n2, b2, midp(b1,b2), c2));


/* lev[i] is just a CONSTANT to construct a Level surface on S^3 which
   is applicable */
```

```
/* XC, YC, ZC, WC are Coordinate accessor functions (MACROS) in LISP */

type_B_Aclause[1]:
 type_B_Clause([xc(N), YC(N), ZC(N), WC(N)],
               [xc(u), yc(u), zc(u)],
               [xc(v), yc(v), zc(v)],
               Level);

/* b[1] : type_b_clause( n(i1), v(j1), u(k1), lev[1])
   b[2] : type_b_clause( n(i2), v(j2), u(k2), lev[2]) */

Type_A_Aclause[1]:
 type_A_Clause([xc(N), YC(N), ZC(N), WC(N)],
               [xc(u), yc(u), zc(u)],
               [xc(v), yc(v), zc(v)],
               Level);

/* a[1] : type_a_clause( n(l1), u(m1), v(p1), lev[3])
   a[2] : type_a_clause( n(l2), u(m2), v(p2), lev[4]) */

/* C_1[1] : type_c1_clause( n(b1), n(b2), v(ai1), v(ai2), lev[5], lev[6])
   C_2[1] : type_c2_clause( n(a1), n(a2), u(bj1), u(bj2), lev[7], lev[8]) */

/* Grind the results... */

print("
Type_A_Aclause[1] :" )$ Grind(Type_A_Aclause[1])$

print("
Type_B_Aclause[1] : ")$ Grind(type_B_Aclause[1])$
```

```
/* bruce r. donald. cspace constraints in 3-dimensions. -*- macsyma -*- */

/* production-version: i.e. produce lisp code...
   xc, yc and zc are baccessor macros for components of vectors */

/* this next section contains the equations for c-surfaces in
 r^3 x s^3. see aim 605, tomas' spatial planning paper for details. */

/* load euler rotation equations */
 if euler_rotation_equations_loaded = true
        then "ok"
        else    (batchload("usrd$:[brd.prod]rotate.mac"),
                 euler_rotation_equations_loaded:true);

/* a vector in r^3: */

xvec : [x, y, z];

ratvars: [x, y, z,
                sin(phi), cos(phi), sin( theta), cos(theta),
                sin(psi), cos(psi) ];

/* a vertex on a , and a(i+1) */

ai :   [xc(ai) , yc(ai), zc(ai)];
aione: [xc(aione), yc(aione), zc(aione)];

/* a vertex on b , and b(j+1) */

bj:    [xc(bj),  yc(bj), zc(bj)];
bjone: [xc(bjone), yc(bjone), zc(bjone)];

/* the normal to a face fi on a */

nfi: [ xc(nfi), yc(nfi), zc(nfi), wc(nfi)];

/* the normal to a face gj on b */

ngj : [xc(ngj), yc(ngj), zc(ngj), wc(ngj)];

/* type a surface. rotate the normal: */

R3_projection(Vec) := [vec[1], vec[2], vec[3]]$

Rot_Nfi: part(Transform(Euler_Inverse,nfi),1);

N_a: R3_Projection(Rot_Nfi);

Inner_Product_Term:  (Rotate_vector(ai) + bj);

A_6: N_a . Xvec - (N_a . Inner_Product_Term);

/* Type B Surface */

N_b: R3_projection(Ngj);

B_6 : N_b . Xvec - (N_b . Inner_Product_Term);

/* Type C surface (!) */


Edge_a: rotate_vector(aione) - rotate_vector(ai);

Edge_b: Bjone - bj;

N_C: Cross( Edge_a, Edge_b);

C_6: N_C . Xvec - (N_c . Inner_Product_Term);
```

```
/* Simplify if possible? */

Simp_1(Exp, Var) := ratsubst(1, sin(var)^2 + cos(var)^2, exp);
simp_3(Exp) := Rat(Simp_1( Simp_1( Simp_1(Exp, Phi), Theta), Psi));

A_5: Simp_3(ratsimp(A_5));
B_5: Simp_3(ratsimp(B_5));
C_5: Simp_3(ratsimp(C_5));

[length(a_5), length(b_5), length(C_5)];

/* Grind the forms here */

Print("
Type_A_Csurface[1] :") $ grind(A_5)$

Print("
Type_B_Csurface[1] :")$  Grind(B_5)$

Print("
Type_C_Csurface[1] :")$  Grind(C_5)$
```

```
/* -*- MODE: MACSYMA -*-
   (HERE ARE THE DEFINITIONS OF TYPE (A) (B) AND (C) C-SURFACES FOR
         THE 6DOF MOVERS PROBLEM, AND TYPE (A) AND (B) APPLICABILITY
         CONSTRAINTS . Output of CSPACE and APPLIC under PRODUCE for
         production run.) */


Type_A_Csurface[1] :

(((XC(NFI)*X-XC(BJ)*XC(NFI))*COS(PHI)+(XC(NFI)*Y-YC(BJ)*XC(NFI))*SIN(PHI))
 *COS(THETA)
 +(-XC(NFI)*Z+ZC(BJ)*XC(NFI))*SIN(THETA)+(YC(NFI)*Y-YC(BJ)*YC(NFI))*COS(PHI)
 +(-YC(NFI)*X+XC(BJ)*YC(NFI))*SIN(PHI)-YC(NFI))
 *COS(PSI)
 +(((-YC(NFI)*X+XC(BJ)*YC(NFI))*COS(PHI)
  +(-YC(NFI)*Y+YC(BJ)*YC(NFI))*SIN(PHI))
  *COS(THETA)
  +(YC(NFI)*Z-ZC(BJ)*YC(NFI))*SIN(THETA)+(XC(NFI)*Y-YC(BJ)*XC(NFI))*COS(PHI)
  +(-XC(NFI)*X+XC(BJ)*XC(NFI))*SIN(PHI)-XC(NFI))
  *SIN(PSI)+(ZC(NFI)*Z-ZC(BJ)*ZC(NFI))*COS(THETA)
 +((ZC(NFI)*X-XC(BJ)*ZC(NFI))*COS(PHI)+(ZC(NFI)*Y-YC(BJ)*ZC(NFI))*SIN(PHI))
  *SIN(THETA)-ZC(AI)*ZC(NFI)-YC(AI)*YC(NFI)-XC(AI)*XC(NFI)$



Type_B_Csurface[1] :


((-XC(AI)*XC(NGJ)*COS(PHI)-XC(AI)*YC(NGJ)*SIN(PHI))*COS(THETA)
 +XC(AI)*ZC(NGJ)*SIN(THETA)-YC(AI)*YC(NGJ)*COS(PHI)+YC(AI)*XC(NGJ)*SIN(PHI))
 *COS(PSI)
 +((YC(AI)*XC(NGJ)*COS(PHI)+YC(AI)*YC(NGJ)*SIN(PHI))*COS(THETA)
  -YC(AI)*ZC(NGJ)*SIN(THETA)-XC(AI)*YC(NGJ)*COS(PHI)+XC(AI)*XC(NGJ)*SIN(PHI))
  *SIN(PSI)-ZC(AI)*ZC(NGJ)*COS(THETA)
 +(-ZC(AI)*XC(NGJ)*COS(PHI)-ZC(AI)*YC(NGJ)*SIN(PHI))*SIN(THETA)
 -YC(NGJ)*COS(PHI)+XC(NGJ)*SIN(PHI)+ZC(NGJ)*Z+YC(NGJ)*Y+XC(NGJ)*X
 -ZC(BJ)*ZC(NGJ)-YC(BJ)*YC(NGJ)-XC(BJ)*XC(NGJ)$

Type_C_Csurface[1] :


(((((XC(AIONE)-XC(AI))*YC(BJONE)+(-XC(AIONE)+XC(AI))*YC(BJ))*Z
 +((-XC(AIONE)+XC(AI))*ZC(BJONE)+(XC(AIONE)-XC(AI))*ZC(BJ))*Y
 +(XC(AIONE)-XC(AI))*YC(BJ)*ZC(BJONE)+(-XC(AIONE)+XC(AI))*ZC(BJ)*YC(BJONE)
 +(-YC(AI)*ZC(AIONE)+ZC(AI)*YC(AIONE))*XC(BJONE)
 +(YC(AI)*ZC(AIONE)-ZC(AI)*YC(AIONE))*XC(BJ))
 *COS(PHI)
 +(((-XC(AIONE)+XC(AI))*XC(BJONE)+(XC(AIONE)-XC(AI))*XC(BJ))*Z
  +((XC(AIONE)-XC(AI))*ZC(BJONE)+(-XC(AIONE)+XC(AI))*ZC(BJ))*X
  +(-XC(AIONE)+XC(AI))*XC(BJ)*ZC(BJONE)
  +(-YC(AI)*ZC(AIONE)+ZC(AI)*YC(AIONE))*YC(BJONE)
  +(XC(AIONE)-XC(AI))*ZC(BJ)*XC(BJONE)
  +(YC(AI)*ZC(AIONE)-ZC(AI)*YC(AIONE))*YC(BJ))
  *SIN(PHI)+(XC(AIONE)-XC(AI))*ZC(BJONE)+(-XC(AIONE)+XC(AI))*ZC(BJ))
 *COS(THETA)
 +(((XC(AIONE)-XC(AI))*XC(BJONE)+(-XC(AIONE)+XC(AI))*XC(BJ))*COS(PHI)
  +((XC(AIONE)-XC(AI))*YC(BJONE)+(-XC(AIONE)+XC(AI))*YC(BJ))*SIN(PHI)
  +((-XC(AIONE)+XC(AI))*XC(BJONE)+(XC(AIONE)-XC(AI))*XC(BJ))*Y
  +((XC(AIONE)-XC(AI))*YC(BJONE)+(-XC(AIONE)+XC(AI))*YC(BJ))*X
  +(YC(AI)*ZC(AIONE)-ZC(AI)*YC(AIONE))*ZC(BJONE)
  +(-XC(AIONE)+XC(AI))*XC(BJ)*YC(BJONE)+(XC(AIONE)-XC(AI))*YC(BJ)*XC(BJONE)
  +(-YC(AI)*ZC(AIONE)+ZC(AI)*YC(AIONE))*ZC(BJ))
  *SIN(THETA)
 +(((-YC(AIONE)+YC(AI))*XC(BJONE)+(YC(AIONE)-YC(AI))*XC(BJ))*Z
  +((YC(AIONE)-YC(AI))*ZC(BJONE)+(-YC(AIONE)+YC(AI))*ZC(BJ))*X
  +(-YC(AIONE)+YC(AI))*XC(BJ)*ZC(BJONE)
  +(XC(AI)*ZC(AIONE)-ZC(AI)*XC(AIONE))*YC(BJONE)
  +(YC(AIONE)-YC(AI))*ZC(BJ)*XC(BJONE)
  +(-XC(AI)*ZC(AIONE)+ZC(AI)*XC(AIONE))*YC(BJ))
```

```
     *COS(PHI)
    +(((-YC(AIONE)+YC(AI))*YC(BJONE)+(YC(AIONE)-YC(AI))*YC(BJ))*Z
     +((YC(AIONE)-YC(AI))*ZC(BJONE)+(-YC(AIONE)+YC(AI))*ZC(BJ))*Y
     +(-YC(AIONE)+YC(AI))*YC(BJ)*ZC(BJONE)+(YC(AIONE)-YC(AI))*ZC(BJ)*YC(BJONE)
     +(-XC(AI)*ZC(AIONE)+ZC(AI)*XC(AIONE))*XC(BJONE)
     +(XC(AI)*ZC(AIONE)-ZC(AI)*XC(AIONE))*XC(BJ))
     *SIN(PHI))
    *COS(PSI)
    +(((((-YC(AIONE)+YC(AI))*YC(BJONE)+(YC(AIONE)-YC(AI))*YC(BJ))*Z
     +((YC(AIONE)-YC(AI))*ZC(BJONE)+(-YC(AIONE)+YC(AI))*ZC(BJ))*Y
     +(-YC(AIONE)+YC(AI))*YC(BJ)*ZC(BJONE)+(YC(AIONE)-YC(AI))*ZC(BJ)*YC(BJONE)
     +(-XC(AI)*ZC(AIONE)+ZC(AI)*XC(AIONE))*XC(BJONE)
     +(XC(AI)*ZC(AIONE)-ZC(AI)*XC(AIONE))*XC(BJ))
     *COS(PHI)
     +(((YC(AIONE)-YC(AI))*XC(BJONE)+(-YC(AIONE)+YC(AI))*XC(BJ))*Z
      +((-YC(AIONE)+YC(AI))*ZC(BJONE)+(YC(AIONE)-YC(AI))*ZC(BJ))*X
      +(YC(AIONE)-YC(AI))*XC(BJ)*ZC(BJONE)
      +(-XC(AI)*ZC(AIONE)+ZC(AI)*XC(AIONE))*YC(BJONE)
      +(-YC(AIONE)+YC(AI))*ZC(BJ)*XC(BJONE)
      +(XC(AI)*ZC(AIONE)-ZC(AI)*XC(AIONE))*YC(BJ))
      *SIN(PHI)+(-YC(AIONE)+YC(AI))*ZC(BJONE)+(YC(AIONE)-YC(AI))*ZC(BJ))
     *COS(THETA)
     +(((-YC(AIONE)+YC(AI))*XC(BJONE)+(YC(AIONE)-YC(AI))*XC(BJ))*COS(PHI)
      +((-YC(AIONE)+YC(AI))*YC(BJONE)+(YC(AIONE)-YC(AI))*YC(BJ))*SIN(PHI)
      +((YC(AIONE)-YC(AI))*XC(BJONE)+(-YC(AIONE)+YC(AI))*XC(BJ))*Y
      +((-YC(AIONE)+YC(AI))*YC(BJONE)+(YC(AIONE)-YC(AI))*YC(BJ))*X
      +(XC(AI)*ZC(AIONE)-ZC(AI)*XC(AIONE))*ZC(BJONE)
      +(YC(AIONE)-YC(AI))*XC(BJ)*YC(BJONE)+(-YC(AIONE)+YC(AI))*YC(BJ)*XC(BJONE)
      +(-XC(AI)*ZC(AIONE)+ZC(AI)*XC(AIONE))*ZC(BJ))
      *SIN(THETA)
     +(((-XC(AIONE)+XC(AI))*XC(BJONE)+(XC(AIONE)-XC(AI))*XC(BJ))*Z
      +((XC(AIONE)-XC(AI))*ZC(BJONE)+(-XC(AIONE)+XC(AI))*ZC(BJ))*X
      +(-XC(AIONE)+XC(AI))*XC(BJ)*ZC(BJONE)
      +(-YC(AI)*ZC(AIONE)+ZC(AI)*YC(AIONE))*YC(BJONE)
      +(XC(AIONE)-XC(AI))*ZC(BJ)*XC(BJONE)
      +(YC(AI)*ZC(AIONE)-ZC(AI)*YC(AIONE))*YC(BJ))
      *COS(PHI)
     +(((-XC(AIONE)+XC(AI))*YC(BJONE)+(XC(AIONE)-XC(AI))*YC(BJ))*Z
      +((XC(AIONE)-XC(AI))*ZC(BJONE)+(-XC(AIONE)+XC(AI))*ZC(BJ))*Y
      +(-XC(AIONE)+XC(AI))*YC(BJ)*ZC(BJONE)+(XC(AIONE)-XC(AI))*ZC(BJ)*YC(BJONE)
      +(YC(AI)*ZC(AIONE)-ZC(AI)*YC(AIONE))*XC(BJONE)
      +(-YC(AI)*ZC(AIONE)+ZC(AI)*YC(AIONE))*XC(BJ))
      *SIN(PHI))
     *SIN(PSI)
    +(((-ZC(AIONE)+ZC(AI))*XC(BJONE)+(ZC(AIONE)-ZC(AI))*XC(BJ))*COS(PHI)
     +((-ZC(AIONE)+ZC(AI))*YC(BJONE)+(ZC(AIONE)-ZC(AI))*YC(BJ))*SIN(PHI)
     +((ZC(AIONE)-ZC(AI))*XC(BJONE)+(-ZC(AIONE)+ZC(AI))*XC(BJ))*Y
     +((-ZC(AIONE)+ZC(AI))*YC(BJONE)+(ZC(AIONE)-ZC(AI))*YC(BJ))*X
     +(-XC(AI)*YC(AIONE)+YC(AI)*XC(AIONE))*ZC(BJONE)
     +(ZC(AIONE)-ZC(AI))*XC(BJ)*YC(BJONE)+(-ZC(AIONE)+ZC(AI))*YC(BJ)*XC(BJONE)
     +(XC(AI)*YC(AIONE)-YC(AI)*XC(AIONE))*ZC(BJ))
     *COS(THETA)
    +((((ZC(AIONE)-ZC(AI))*YC(BJONE)+(-ZC(AIONE)+ZC(AI))*YC(BJ))*Z
     +((-ZC(AIONE)+ZC(AI))*ZC(BJONE)+(ZC(AIONE)-ZC(AI))*ZC(BJ))*Y
     +(ZC(AIONE)-ZC(AI))*YC(BJ)*ZC(BJONE)+(-ZC(AIONE)+ZC(AI))*ZC(BJ)*YC(BJONE)
     +(-XC(AI)*YC(AIONE)+YC(AI)*XC(AIONE))*XC(BJONE)
     +(XC(AI)*YC(AIONE)-YC(AI)*XC(AIONE))*XC(BJ))
     *COS(PHI)
     +(((-ZC(AIONE)+ZC(AI))*XC(BJONE)+(ZC(AIONE)-ZC(AI))*XC(BJ))*Z
      +((ZC(AIONE)-ZC(AI))*ZC(BJONE)+(-ZC(AIONE)+ZC(AI))*ZC(BJ))*X
      +(-ZC(AIONE)+ZC(AI))*XC(BJ)*ZC(BJONE)
      +(-XC(AI)*YC(AIONE)+YC(AI)*XC(AIONE))*YC(BJONE)
      +(ZC(AIONE)-ZC(AI))*ZC(BJ)*XC(BJONE)
      +(XC(AI)*YC(AIONE)-YC(AI)*XC(AIONE))*YC(BJ))
      *SIN(PHI)+(ZC(AIONE)-ZC(AI))*ZC(BJONE)+(-ZC(AIONE)+ZC(AI))*ZC(BJ))
     *SIN(THETA)$

/* Herer are the Applicability constraints */
```

```
Type_A_Aclause[1] :

(((XC(U)-XC(V))*XC(N)*COS(PHI)+(YC(U)-YC(V))*XC(N)*SIN(PHI))*COS(THETA)
 +(-ZC(U)+ZC(V))*XC(N)*SIN(THETA)+(YC(U)-YC(V))*YC(N)*COS(PHI)
 +(-XC(U)+XC(V))*YC(N)*SIN(PHI))
 *COS(PSI)
 +(((-XC(U)+XC(V))*YC(N)*COS(PHI)+(-YC(U)+YC(V))*YC(N)*SIN(PHI))*COS(THETA)
  +(ZC(U)-ZC(V))*YC(N)*SIN(THETA)+(YC(U)-YC(V))*XC(N)*COS(PHI)
  +(-XC(U)+XC(V))*XC(N)*SIN(PHI))
  *SIN(PSI)+(ZC(U)-ZC(V))*ZC(N)*COS(THETA)
 +((XC(U)-XC(V))*ZC(N)*COS(PHI)+(YC(U)-YC(V))*ZC(N)*SIN(PHI))*SIN(THETA)-LEVEL$


Type_B_Aclause[1] :


 (((-XC(N)*XC(V)+XC(N)*XC(U))*COS(PHI)+(-YC(N)*XC(V)+YC(N)*XC(U))*SIN(PHI))
  *COS(THETA)
  +(ZC(N)*XC(V)-ZC(N)*XC(U))*SIN(THETA)+(-YC(N)*YC(V)+YC(N)*YC(U))*COS(PHI)
  +(XC(N)*YC(V)-XC(N)*YC(U))*SIN(PHI))
  *COS(PSI)
  +(((XC(N)*YC(V)-XC(N)*YC(U))*COS(PHI)+(YC(N)*YC(V)-YC(N)*YC(U))*SIN(PHI))
   *COS(THETA)
   +(-ZC(N)*YC(V)+ZC(N)*YC(U))*SIN(THETA)+(-YC(N)*XC(V)+YC(N)*XC(U))*COS(PHI)
   +(XC(N)*XC(V)-XC(N)*XC(U))*SIN(PHI))
   *SIN(PSI)+(-ZC(N)*ZC(V)+ZC(N)*ZC(U))*COS(THETA)
  +((-XC(N)*ZC(V)+XC(N)*ZC(U))*COS(PHI)+(-YC(N)*ZC(V)+YC(N)*ZC(U))*SIN(PHI))
   *SIN(THETA)-LEVEL$
```

```
/* -*- mode: macsyma -*-
 (Bruce Donald. Here we Express a Constraint (C-surface or
   Acf) in Canonical Linear Form and Canonical Trig form.

      Given a CONSTRAINT which is either a
      C-surface or an ACF (applicability clause function)
      and a variable (VAR) we solve for the variable) */

Load_up():=                                  /* load necessary files */
block([],                                    /* for solving */
      if ALL_DEFS_LOADED = TRUE then "OK"
         else batchload("usrd$:[brd.prod]defabc.mac"),
      batchload([intabc,mac]),
      batchload("usrd$:[brd.prod]analyze2.mac"),
      RatVars: [x, y, z,
              sin(phi), cos(phi), sin( theta), cos(theta),
              sin(psi), cos(psi) ],
      Angles: [Phi, Psi, theta])$

Load_up()$

/* give us an "explicit" tangent space */

s(var) := sin(var)$
c(var) := cos(var)$
Build_Manifold()$                            /* Rebuild Manifold */

solve_for_angle(exp,var):=                   /* Solve for COS(var) */
block([Rats, R1, R2, R3],
      rats: ratvars,
      Ratvars: [c(var), s(var)],
      print("Simplifying..."),
      R1: IsolateN(exp, Ratvars),
      Print("Eliminating ", s(var),"..."),
      R2: Eliminate[1](R1, Var),
      Ldisp(R2),
      Print("Solving for ", c(var),"..."),
      R3: solve(R2,c(var)),
      display_and_grind(R3),
      ratvars:rats,
      r3)$

Solve_for_X(Exp, Xvar):=                     /* Solve for Any Var */
Block([Rats, r1, R2],
      Rats: ratvars,
      Ratvars: [Xvar],
      print("Simplifying..."),
      R1: IsolateN(Exp, Ratvars),
      print(" Solving for ", Xvar,"..."),
      r2: solve(R1, Xvar),
      Display_and_grind(r2),
      ratvars: rats,
      R2)$

Solve_test():=                               /* Test the Solution Routines */
block([],
      grind(solve_test),
      grind:false,
      Solve_for_angle(Cs1, Phi),
      Solve_for_X(CS1, X),

      /* SOlve a type (a) surface for PSI and Y */

      AS1: SinCos_to_CS(type_A_CSurface[1]),

      Solve_for_angle(AS1, Psi),
      Solve_for_X(AS1, y),

      /* SOlve a type (B) surface for PSI and Y */
```

```
        BS1: SinCos_to_CS(type_B_CSurface[1]),

        Solve_for_angle(BS1, Psi),
        Solve_for_X(BS1, y),

        Notify()
        )$

/* CExpress expresses things Canonically */

CExpress(Exp, Type) :=
 Block([],
         If Type = C_Surface then
          (Print("Canonical Linear Form ..."),
                Canonical_linear_form(Exp)),
         Print("Solving for Angles..."),
         for Var in Angles do
           (Solve_for_angle(Exp, Var)))$

/* Here we Do the Expression. Now to get Ground forms, just change GRIND, etc. */

DO_CExpress():=
Block([],
        Grind: false,                    /* Here it is.. */
        kill(labels),
        CExpress(type_A_Csurface[1], 'C_Surface),
        CExpress(type_B_Csurface[1], 'C_Surface),
        CExpress(type_C_Csurface[1], 'C_Surface),
        CExpress(type_A_Aclause[1],  'ACF),
        CExpress(type_B_AClause[1],  'ACF))$
```

```
/* -*- Macsyma -*- BRD@OZ (Bruce  R. Donald).
       Sove intersections of C_surfaces and A_clauses.
       for the 6dof movers problem. */

Angles: [theta, Phi, Psi]$


/* allow some simplification into C and S terms */

SinCos_to_CS (exp) :=
Block([E],
       E: exp,
       for Var in Angles Do
       E: ratsubst(s[var], sin(var),
                   ratsubst(c[var], cos(var), E)),
       rat(E))$

/* permit the inverse */

CS_to_SinCos (exp) :=
Block([E],
       E: exp,
       for Var in Angles Do
       E: ratsubst(sin(var), s[var],
                   ratsubst(cos(var),c[var], E)),
       E)$

/* The Tangent Space Manifold */

/* DO it yourself, pal!  batchload([defabc,mac, "usrd$", brd]) */

/* Short form functions */

s(var) := s[var]$
c(var) := c[var]$

/* SImplify if possible? use the s/c[var] form though. */

Simp_1(Exp, Var) := ratsubst(1, s(var)^2 + c(var)^2, exp)$
simp_3(Exp) := Rat(
   Simp_1( Simp_1( Simp_1(Exp, Phi), Theta), Psi))$

manifold(var):= s(var)^2 + c(var)^2 = 1$

Build_manifold() :=
Block([],
       Man[theta]: manifold(theta),
       Man[Phi] : Manifold(phi),
       Man[Psi] : Manifold(psi))$

Build_manifold()$

/* Eliminate[1] eliminates the dual trig term. S^1 is parameterized by
   one variable (var). Eliminates SIN(VAR) from EXP */

eliminate[1](exp, var) :=
   block([],
         Temp1: rat(part(
                   eliminate([exp, Man[var]], [s(var)]),1))
          )$

/* Eliminate[2] calls eliminate[1] twice, and eliminates the resultants.
   Hence the intersection of two level surfaces on S^3 is
   parameterized by a one param. family. Eliminates Sin/cos Var1/Var2
   from Exp. */

USE_CS_FORM: True$

eliminate[2](exp1, exp2, var1, var2) :=
```

```
block( [R1, R2, R3, R4, R5,   r6,                    /* Temp results */
        Lr1, Lr2, Lr3, Lr4, Lr5, lr6],               /* Their Labels */
 RatVars: [x, y, z,
           s(phi), c(phi), s( theta), c(theta),
           s(psi), c(psi) ],
         if Use_CS_FORM then
            (exp1: SinCos_TO_CS(exp1),
             exp2: SinCos_To_CS(exp2)),
        print("Eliminate ",s(var1)," from expression 1:"),
        r1: eliminate[1](exp1, var1),
        Lr1:ldisp(r1),
          print("Eliminate ",s(var1)," from expression 2:"),
          r2: eliminate[1](exp2, var1),
          Lr2:ldisp(r2),
        print("Eliminate ",c(var1)," from ",append(LR1, LR2)),
        r3: rat(eliminate([r1,r2],[c(var1)])),
        Lr3: Ldisp(R3),
         print("Eliminate ",s(var2)," from ",Lr3),
         R4: eliminate[1](R3, var2),
         Lr4: Ldisp(R4),
        Print(" Solve ", lr4," for ", c(var2)),
        R5: rat(Solve(R4, c(var2))),
        Lr5: Ldisp(R5),
         print("Finally, solve ",Lr1," for ",c(var1)),
         R6: rat(solve(R1, c(var1))),
         Lr6: ldisp(R6),
        Append(lr2,lr2,lr3,lr4,lr5,lr6))$

Test():=
 eliminate[2](type_A_Aclause[1],
              type_A_Aclause[2],
              Phi,
              Psi)$
```

```
 /*     -*- Mode: macsyma -*-
  (File to Run Production of  3d Space equations.
   Bruce R. Donald, MIT AI LAB ) */

/* Define the type ABC constraints. */

Define_ABC():=
block([],
    .     batch("USRD$:[brd.prod]Cspace.mac"))$

/* Define the appllicability Constraints */

Define_Applic():=
Block([],
        Batch("USRD$:[brd.prod]Applic.mac"))$

/* Define Both */

Produce_Defs():=
block([],
        Writefile("usrd$:[brd.prod]Produce.log"),
        Batch("usrd$:[brd.prod]Rotate.mac"),
        Define_ABC(),
        Define_Applic(),
        Closefile(),
        Notify())$

Cold_Restart():=
block([],
        batchload("sys$login:utils.mac"),
        batchload("usrd$:[brd.prod]produce.mac"),
        Produce_defs())$

/* Here's a function to save labels for you. */

Save_labels(file):=
block([],
        ?Open_output_file(file),
        for Label in Append( reverse(labels(e)), reverse(labels(d)))
        DO
         (?Grind_TO_FILE(Label), Print(Label)),
        ?Close_Output_file())$

/* Produce the EXPRESS file, parsing into solutions and coefficients. */

Produce_Express():=
block([],
        writefile("usrd$:[brd.prod]Express.log"),
        batch("usrd$:[brd.prod]Express.mac"),
        DO_CExpress(),
        Print(" Saving Labels in LSP file..."),
        Save_Labels("USRD$:[brd.prod]EXPRESS.LSP"),
        Notify())$

/* this function produces EVERYTHING. */

Produce_ALL():=
BLOCK([],
        Produce_Defs(),
        ALL_DEFS_LOADED: TRUE,
        Kill(Nfi, Ngj, Ai, Aione, Bj, Bjone),
        Type_A_Csurface[1] : A_6,
        Type_B_Csurface[1]: B_5,
        Type_C_Csurface[1]: C_6,
        Produce_express(),
        closefile())$
```

```
/* Bruce R. Donald (BRD@OZ) Euler Rotations for Macsyma. See -*- MACSYMA -*-
   Paul, p 45 */

c(angle):= cos(angle)$
s(angle):= sin(angle)$

Rot_z_psi:
  matrix(
[ c(psi), -s(psi), 0, 0],
[ s(psi),  c(psi), 0, 0],
[ 0,       0,      1, 0],
[ 0,       0,      0, 1]);

Rot_y_theta: matrix(
[ c(theta), 0, s(theta), 0],
[ 0,        1, 0,        0],
[-s(theta), 0, c(theta), 0],
[ 0,        0, 0,        1]);

Rot_Z_Phi: Matrix(
[ c(phi), -s(phi), 0, 0],
[ s(phi),  c(phi), 0, 0],
[ 0,       0,      1, 0],
[ 0,       0,      0, 1]);

Euler_temp: Rot_Y_Theta . Rot_Z_Psi;

Euler_matrix: Rot_Z_Phi . Euler_temp;

homogenize(X):= [x[1], x[2], x[3], 1];

UnHomogenize(X) := [x[1]/x[4], x[2]/x[4], x[3]/x[4]];

Rotate_vector(X) := UnHomogenize( Euler_Matrix . Homogenize(X)) ;

/* Cross Product */
Cross(A,B) :=
 [ (a[2]*b[3] - a[3]*b[2]),
   (a[3]*b[1] - a[1]*b[3]),
   (a[1]*b[2] - a[2]*b[1])];

/* SImplify if possible? */

Simp_1(Exp, Var) := ratsubst(1, sin(var)^2 + cos(var)^2, exp);
simp_3(Exp) := Rat(
   Simp_1( Simp_1( Simp_1(Exp, Phi), Theta), Psi));

/* General Transformation function. A 4-vector is assumed to be a plane,
   and a 3-vector a 3-vector. hence the 3-rector is ROTATED and a 4-vector
   plane is also rotated... See PAUL */

Transform(Transformation_matrix, X) :=
   Block([Hom, Trans,ans],
   Ans: "Whoops!",
   if length(X) = 4 then
         Ans: x .Transformation_matrix
         Else
         if length (x) = 3 then
         Ans:Unhomogenize(Transformation_matrix . Homogenize (X))
           Else Print(" But ",x," Must be a 3-vector or 4-Vector!"),
         Ans);


/* Now Compute Inverse of the Euler Transformation */

E_Adj: simp_3(rat (adjoint( Euler_matrix)));
```

References

Baer, A., Eastman, C., and Henrion, M. "Geometric Modeling: A survey," *Computer-Aided Design* **11**, 5 (1979).

Binford, Thomas "Visual Perception by Computer," *IEEE Systems Science and Cybernetics Conference* , Miami, 1971.

Brady, J. M. *et al. Robot Motion: Planning and Control* , MIT Press, Cambridge, MA, 1983.

Brady, J. M. "Criteria for Representations of Shape," *Human and Machine Vision* eds. Rosenfeld A., and Beck J., 1982.

Brady, J. M. "Smoothed Local Symmetries and Local Frame Propagation," *Proc. Patt. Rec. and Im. Proc.* , Las Vegas, 1982b.

Brooks, Rodney A. "Symbolic Error Analysis and Robot Programming," *International Journal of Robotics Research* **1, no. 1** (1982).

Brooks, Rodney A. "Solving the Find-Path Problem by Good Representation of Free Space," *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-13** (1983a).

Brooks, Rodney, A. "Find-Path for a PUMA-Class Robot," *AAAI* , Washington, DC, 1983b.

Brooks, Rodney A. and Lozano-Pérez, Tomás "A Subdivision Algorithm in Configuration Space for Findpath with Rotations," *International Joint Conference on Artificial Intelligence* , Karlsruhe, Germany, 1983.

Brou, Philippe Finding the Orientation of Objects in Vector Maps, Ph.D Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1983.

Burke, G., *et al.* "The NIL Reference Manual," Laboratory for Computer Science, Massachusetts Institute of Technology, 1983.

Canny, John "On Detecting Collisions Between Polyhedra," *European Conference on Artificial Intelligence* , Pisa, Italy, *To be presented October, 1984.*

Chatila, Raja Système de Navigation pour un Robot Mobile Autonome: Modelisation et Processus Décisionnels, Ph.D. Thesis, L'Université Paul Sabatier de Toulouse, 1981.

Chazelle, Bernard "Computational Geometry and Convexity," Department of Computer Science, Carnegie-Mellon University, CMU-CS-80-150, 1980.

Dobkin, David P. and Kirkpatrick, David G. "Fast Detection of Polyhedral Intersections," Department of Electrical ENgineering and Computer Science, Princeton University, 1980.

Donald, Bruce R. "The Mover's Problem in Automated Structural Design," *Proceedings, Harvard Computer Graphics Conference* , Cambridge, July, 1983b.

Donald, Bruce R. "Hypothesizing Channels Through Free-Space in Solving the Findpath Problem," Artificial Intelligence Laboratory, Massachusetts Institute of Technology, A.I. Memo 736, June, 1983a.

Donald, Bruce R. Local and Global Techniques for Motion Planning, S.M. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 10, 1984.

Drysdale, Robert L. Generalized Voronoi Digrams and Geometric Searching, Department of Computer Science, Stanford University, 1979.

Erdmann, Michael On Motion Planning with Uncertainty, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, August, 1984.

Foley, J. D. and van Dam, A. *Principles of Interactive Computer Graphics* , Addison-Wesley, Reading, Mass., 1982.

Forbus, Kenneth D. "A Study of Qualitative and Geometric Knowledge in Reasoning about Motion," Massachusetts Institute of Technology Artificial Intelligence Laboratory, AI-TR-615, 1981.

Giblin, P. J. *Graphs, Surfaces, and Homology* , Chapman and Hall, London, 1977.

Gouzenes, Laurent "Strategies for Solving Collision-Free Trajectories Problems for Mobile and Manipulator Robots," Laboratoire d'Automatique et d'Analyse des Systemes du CNRS, Toulouse, France, 1983.

Grünbaum *Convex Polytopes* , Interscience Publishers, London, 1967.

Hamilton, W. R. *Elements of Quaternions* , Chelsea Publishing Co., New York, 1969.

Hirsch, M. *Differential Topology* , Springer-Verlag, New York, 1976.

Hocking, J. and Young, G. *Topology* , Addison-Wesley, Reading, Mass., 1961.

Hopcroft, J., Joseph, D., and Whitesides, S. "On The Movement of Robot Arms In 2-Dimensional Regions," Cornell University, TR 82-486, 1982.

Hopcroft, J. and Wilfong, G. "On the Motion of Objects in Contact," Cornell University, Computer Science Department, TR 84-602, 1984.

Hopcroft, J. and Wilfong, G. "On the Motion of Objects in Contact," Cornell University, Computer Science Department, TR 84-602, 1984.

Kalay, Yehuda E. "Determining the Spatial Containment of a Point in General Polyhedra," *Computer Graphics and Image Processing* **Vol. 19** (1982), 303-334.

Kane, T.R. and Levinson, D. A. "Successive Finite Rotations," *Journal of Applied Mechanics* **5** (1978).

LCS Mathlab Group "MACSYMA reference Manual, Volumes I-I," The Mathlab Group, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983.

Lozano-Pérez, Tomás "Spatial Planning: A Configuration Space Approach," *IEEE Transactions on Computers* **C-32** (February, 1983).

____ "Automatic Planning of Manipulator Transfer Movements," *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-11, No. 10** (1981).

Lozano-Pérez, T., Mason, M., and Taylor, R. "Automatic Synthesis of Fine-Motion Strategies for Robots," Massachusetts Institute of Technology Artificial Intelligence Laboratory, A.I. Memo 759, 1983.

Lozano-Pérez, T. and Wesley, M. A. "An Algorithm for Planning Collision-Free Paths among Polyhedral Obstacles," *Communications of the ACM* **22**, 10 (1979).

Mason, M. T. "Compliance and Force Control for Computer-Controlled Manipulators," *SMC-6* (1981).

Massey, Wm. S. *Algebraic Topology* , Springer-Verlag, New York, 1967.

Moravec, H. P. "Visual Mapping by a Robot Rover," *Proceedings Sixth International Joint Conference on Artificial Intelligence* , Tokyo, Japan, 1979.

Nguyen, Van-Duc "The Find-Path Problem in the Plane," Artificial Intelligence Laboratory, Massachusetts Institute of Technology, A.I. Memo 760, 1983.

Nievergelt J. and Preparata, F. "Plane-Sweep Algorithms for Intersecting Geometric Figures," *Communications of the ACM* **25**, 10 (1982).

Nilsson, Nils *Principles of Artificial Intelligence* , Tioga Publishing Co., Palo-Alto, 1980.

Ó'Dúnlaing, C. and Yap, C. "The Voronoi Diagram Method of Motion Planning: I. The Case of a Disc," Courant Institute of Mathematical Sciences, 1982.

Ó'Dúnlaing C., Sharir, M, C. and Yap, C. "Retraction: A New Approach to Motion Planning," Courant Institute of Mathematical Sciences, 1982.

O'Neill, B. *Elementary Differential Geometry* , Academic Press, New York, 1966.

Paul, L. *Robot Manipulation* , MIT press, Cambridge, MA, 1981.

Popplestone, R., Ambler, A., and Bellos, I. "An Interpreter for Describing Assemblies," *Artificial Intelligence* **14, no. 1** (1980).

Preparata, F. and Hong, S. "Convex Hulls of Finite Sets of Points in Two and Three Dimensions," *Communications of the ACM* **23**, 3 (1977).

Preparata, F. and Muller, D. "Finding the Intersection of n Half-Spaces in Time $O(n \log n)$," Coordinated Science Laboratory, University of Illinois, Urbana, Ill., R-803, 1977.

Reif, John H. "The Complexity of the Movers Problem and Generalizations," *Proceedings, 20th Symposium on the Foundations of Computer Science* , 1979.

Requicha, A. A. G. "Representation of Rigid Solids: Theory, Methods, and Systems," *ACM Computing Surveys* **12**, 4 (1980).

Schwartz, Jacob and Sharir, Micha "On the Piano Movers Problem, I: The case of a Two-dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers," Courant Institute of Mathematical Sciences, Report No. 39, 1981.

Schwartz, Jacob and Sharir, Micha "On the Piano Movers Problem, II: General Techniques for Computing Topological Properties of Real Algebraic Manifolds," Courant Institute of Mathematical Sciences, Report No. 41, 1982a.

Schwartz, Jacob and Sharir, Micha "On the Piano Movers Problem, III: Coordinating the Motion of Several Independent Bodies: The Special Case of Circular Bodies Moving Amidst Polygonal Barriers," Courant Institute of Mathematical Sciences, 1982b.

Sechrest, Stuart and Greenberg, Donald "A Visible Polygon Reconstruction Algorithm," *ACM Transactions on Graphics* **Vol. 1**, No. 1 (1982), 25-42.

Spivak, M. *A Comprehensive Introduction to Differential Geometry* , Publish or Perish, Inc, Berkeley, CA, 1979.

Sutherland, Sproull, *et al.* "A Characterization Of Ten Hidden-Surface Algorithms," *Acm Computing Surveys* **6**, 1 (1974).

Symon, K. R. *Mechanics* , Addison-Wesely, Reading, Mass., 1971.

Udupa, S. Collision Detection and Avoidance in Computer-Controlled Manipulators, Ph.D Thesis, Department of Electrical Enginnering, California Institute of Technology, 1977.

Widdoes, C. "A Heuristic Collision Avoider for the Stanford Robot Arm," Stanford Artificial Intelligence Laboratory, 1974.

Wingham, M. Planning How to Grasp Objects in a Cluttered Environment, M. Phil. Thesis, Department of Artificial Intelligence, Edinburgh, 1977.

Winston, P. H. and Horn, B. K. P. *LISP* , Addison-Wesely, Reading, Mass., 1981.

END

FILMED

3-85

DTIC